



**UNIVERSITY OF
BIRMINGHAM**

School of Computer Science

BLOCKCHAIN-BASED MEDICAL DATA MANAGEMENT SYSTEM

MSc Advanced Computer Science

DEEPTHI SARVAMANGALA MOULI

Student ID: 2801976

Under the supervision of

Wendy Yanez Pazmino

September 2024-25

Abstract

Electronic Medical Record (EMR) systems are widely used, but their current designs face issues such as maintaining tamper-evident integrity and a reliable chronology of clinical data. This is true when insiders or administrators are considered trustworthy. Traditional database-only solutions allow undetected retroactive edits, replays, or back-dating. These practices can undermine medical, legal compliance, and ethical accountability, which directly threatens patient safety and weakens trust in healthcare systems.

This project solves this issue by proposing a prototype EMR system that embeds a blockchain within the application to secure the entire clinical record. All EMR data is stored as SHA-256 hash-linked blocks mined with Proof-of-Work (difficulty = 5). User management and appointment metadata are kept in MySQL. The system is built in Java (using the Swing client-server model), the system supports role-specific workflows. Doctors can accept/reject appointment requests, create and update entries. Patients can book appointments and view reports. Laboratories can upload results after validating them using an OTP sent via email. Third parties receive read-only access through an Appointment ID and date of birth (DOB).

Evaluation shows that changes, replays, and back-dating attempts are detected through broken hash links, missing PoW prefix compliance, and timestamp anomalies. Storage growth is linear, and mining overhead remains within interactive limits on standard hardware. The prototype prioritises integrity over availability, creating a new mini-chain upon restart. Its novelty lies in extending blockchain EMRs beyond the traditional doctor-patient model. This is done by adding OTP-controlled lab uploads and secure third-party access, showing how cryptographic verification can be included in practical clinical workflows.

Keywords: Electronic Medical Records, Blockchain, Proof-of-Work, SHA-256, Java, OTP, Tamper-evidence, Integrity, Security

Acknowledgement

I want to thank my professor, Wendy Yanez Pazmino, for her valuable guidance, support, and encouragement throughout this project. Her knowledge, helpful feedback, and ongoing motivation have been crucial to completing this work.

I also appreciate my inspector, Luca Arnaboldi, for his thoughtful feedback during the presentation and demonstration. His insights helped improve my project.

I would like to express gratitude to my parents. Their steady support, sacrifices, and unconditional love have been the foundation of my academic journey. Their belief in me has been a constant source of strength and determination.

I dedicate this work to them in recognition of their significant role in my personal and academic growth.

Table of Contents

1 Introduction	7
1.1 Problem Statement	7
1.2 Aims	7
1.3 Motivation	7
1.4 Scope and Contributions	8
1.5 Structure of the report	8
2 Background	8
2.1 Electronic Medical Records (EMRs) in healthcare	8
2.2 Challenges in current EMR Systems	9
2.3 Blockchain fundamentals	9
2.4 Blockchain in Healthcare	10
2.5 Regulatory and Ethical Considerations	10
2.6 Positioning for this study	11
3 Literature Review	11
3.1 Blockchain for healthcare data	11
3.2 Integrity and auditability	12
3.3 Consent and access models	12
3.4 Storage placement trade-offs	13
3.5 Gap and positioning	13
4 Problem Analysis	14
4.1 Insider threats	14
4.2 Weaknesses of Hybrid Blockchain-Database Models	14
4.3 Limitations of Traditional Audit and Cryptographic Methods	14
4.4 Usability vs Security Challenges	15
4.5 Missing Role Diversity	15
4.6 Need for Integrity-First EMRs	15
5 Project Specification	15
5.1 Software methodology	15
5.2 Requirement gathering	16
5.3 System requirements	16
5.5 Non-functional requirements (NFRs)	18
5.6 Constraints and assumptions	19
5.7 Roles and permissions matrix	20
5.8 Out-of-scope	20
5.9 Project Management	21
6 Solution Design	21
6.1 Architecture overview	21
6.2 Data placement and models	21
6.3 Block structure and mining	22
6.4 Socket protocol	22
6.5 Access control model	23
6.6 Rationales	23
6.7 Verification procedure:	26
7 Implementation	26
7.1 Run and deploy	26
7.2 Technology stack	27
7.3 Server components	27
7.4 Blockchain core	28
7.6 Database access and schema	29
7.7 Restart policy and persistence	29

7.8 Error handling and logging.....	30
8 Evaluation and results	30
8.1 Objectives and approach	30
8.2 Testbed	31
8.2.1 How we manually verify	31
8.3 Risk-based testing	31
8.4 Functional correctness.....	32
8.5 Link-and-prefix Integrity tests (manual checks).....	33
8.6 Backup/restore integrity	33
8.7 Performance (PoW and I/O)	33
8.8 Comparative positioning	35
8.9 Threats to validity	35
8.10 Lessons learnt	36
9 Summary and Conclusion	36
Appendix	39
Appendix A	39
Appendix B	41
Appendix C	42
Appendix D	43
Appendix E	48
Appendix F	51
Appendix G	52
Appendix H.....	54
Appendix I	55
Appendix J.....	58
Appendix K.....	60
Appendix L	61
Appendix M.....	63

Table of Figures

• FR-8 Patient read: The patient can view their own EMR in read-only mode.	
Figure 1: Use-case diagram of 4 modules	17
• Blockchain ledger implemented as a JSON file: Full EMR content stored as SHA-256 hash-linked blocks, mined with Proof-of-Work at difficulty 5, which means five leading hexadecimal zeros.	
Figure 2: End-to-end data path and storage split.....	21
Figure 3: Block linkage and Proof-of-Work structure	22
Figure 4: Class diagram of all implemented modules.....	23
Figure 5: Blockchain core code	28
Figure 6: OTP mail dispatch	29
Figure 7: JDBC connection in Dbconnect.java.	
Figure 8: Runtime endpoints	
in ips.java.....	29
Figure 9: Blockchain block objects in JSON showing tamper-evident links via previousHash and hash fields.....	30
Figure 10: Project Gantt Chart.....	51
Figure 11: Jira Scrum Board (Sprint Management – Work-in-Progress).....	51
Figure 12: Sequence Diagram of Doctor Write Path	52
Figure 13: Evidence of Blockchain Integrity	60
Figure 14: Appointment booking screen	
Figure 15: Appointment successfully created.....	61
Figure 16: Doctor dashboard-pending appointment consultation screen	
Figure 17: Doctor	
Figure 18: Lab technician login with OTP validation	
Figure 19: Guest login screen....	62

Figure 20: Patient report view	Figure 21: Blockchain server console ...	62
Figure 22: JSON ledger showing hash connection and HeidiSQL new chain formation	Figure 23: Appointment metadata stored in	62

List of Tables

Table 1: Roles and permission matrix	20
Table 2: Block fields and descriptions	22
Table 3: Risk Assessment and Validation Strategy	32
Table 4: Some Representative Functional Correctness Test Cases	33
Table 5: Prior hybrids vs this prototype	42
Table 6: Persona 1- Patient.....	44
Table 7: Persona 2- Doctor.....	45
Table 8: Persona 3- Friend/Caretaker (Third-party access)	46
Table 9: Persona 4- Lab Technician	46
Table 10: WCAG 2.1 (AA) - Perceivable	48
Table 11: WCAG 2.1 (AA) - Operable	48
Table 12: WCAG 2.1 (AA) - Understandable.....	48
Table 13: WCAG 2.1 (AA) - Robust	49
Table 14: Requirements for Children.....	49
Table 15: Requirements for Seniors.....	49
Table 16: Socket Protocol Message Contract.....	53
Table 17: Comparison of Storage and Consensus Options.....	54
Table 18: UI Swing components used.....	55
Table 19: Extended Functional Correctness Scenarios	59
Table 20: Tampering Experiments on Ledger Integrity	60

1 Introduction

Electronic Medical Record (EMR) systems are now essential in modern healthcare, allowing documentation of patient histories, supporting clinical decision-making, and providing legal and ethical responsibility. However, their usefulness relies not just on storing but also on keeping that data reliable over time. Any unnoticed changes can have serious medical, legal, and ethical effects [1], [2]

Traditional database-centric solutions are vulnerable as they trust administrators and insiders, allowing undetected retroactive edits, replays, and back-dating. Existing hybrid blockchain-database solutions store only metadata or record hashes on-chain [3], [4], [5]. They keep the main clinical content in traditional database storage [6]. This separation still allows tampering.

This project creates a lightweight EMR prototype where clinical entry is recorded as a SHA-256 hash-linked block, and Proof-of-Work with a difficulty of 5. Unlike designs that only use metadata, the full EMR content is on-chain for straightforward verification. The prototype supports role-based workflows for doctors, patients, lab technicians, and third parties while staying responsive for interactive clinical use.

1.1 Problem Statement

Existing EMR systems depend on policy enforcement and administrator trust. This makes them vulnerable to undetected manipulation. A stronger, tamper-evident approach is necessary to ensure that records stay cryptographically verifiable, even under insider threats.

1.2 Aims

- Store all clinical content on a hash-linked ledger with PoW at difficulty 5.
- Provide role-specific flows: doctors (create/update), patients (register/log in, book appointments, view), lab technician (write after an email OTP challenge), and third parties (read-only with Appointment ID + date of birth).
- Conduct targeted tests to show that edits, replays, and back-dating are detected while keeping the system usable.

1.3 Motivation

Most prior work focuses on multi-institution availability and interoperability. However, it pays less attention to lightweight, integrity-first designs for local clinical use [4], [7], [8].

This prototype fills that gap by showing how a focused prototype can secure end-to-end workflows with minimal infrastructure.

1.4 Scope and Contributions

In Scope (Functional):

On-chain EMR storage holds all clinical records. There are role-specific processes for doctors, patients, labs (with OTP access), and third parties who have read-only access. Metadata like user details and appointments stays off-chain.

In Scope (Non-functional):

Prioritises integrity over availability by implementing tamper evidence through hash links, proof of work, and timestamps. The design detects the edits made in the past, replays, and back-dates while maintaining an interactive performance.

The novelty of this project lies in extending blockchain EMRs beyond the typical patient-doctor model. The system stores complete EMRs on-chain while metadata such as user management and appointments stay off-chain. It also supports OTP-controlled uploads by lab technicians and limits third-party viewing based on Appointment ID and Date of Birth. This integration of different roles into a single blockchain-backed system has not been addressed in previous prototypes. It shows that tamper-proof integrity can coexist with real-world clinical workflows.

1.5 Structure of the report

Reviewing related work is covered in Chapter 2, requirements are outlined in Chapter 3, design and implementation are covered in Chapters 4–5, evaluation is presented in Chapter 6, limitations and future scope are outlined in Chapters 7–8, and the conclusion is provided in Chapter 9. Supporting information is provided in the appendices.

2 Background

2.1 Electronic Medical Records (EMRs) in healthcare

Electronic Medical Records (EMRs) are now important in modern healthcare as they give clinicians organised patient histories, help make diagnostic decisions, and promote clear communication between departments [8]. Unlike traditional paper records, EMRs cut down on redundancy, improve access, and allow for the integration of laboratory, imaging, and pharmacy data. However, the reliability of EMRs relies not just on storage but also on

keeping the information secure over time. Any unnoticed change in a patient's record can directly impact treatment accuracy, legal compliance, and ethical responsibility.

The use of EMRs has increased worldwide because of digitisation efforts and government policies. For example, in the United States, the Health Information Technology for Economic and Clinical Health (HITECH) Act urged healthcare providers to move from paper records to electronic ones. Similarly, in Europe and Asia, digital healthcare plans have highlighted EMRs as the base for connected and patient-focused care [2]. However, technical and security issues still prevent universal trust in EMRs.

2.2 Challenges in current EMR Systems

While EMRs improve efficiency, they face several vulnerabilities like:

- **Insider threats:** Traditional database systems assume that administrators and high-privilege users will act honestly, raising the risk of undetected changes, replays, or backdating [9].
- **Audit limitations:** Most EMRs depend on policy-based logs, like append-only tables and database triggers. However, logs within the same trust domain can be changed or deleted, which undermines accountability [9].
- **Interoperability:** Different hospitals and clinics often use EMR systems that do not work together, making data exchange difficult [8].
- **Privacy and compliance:** Regulations like HIPAA and GDPR impose strict rules for protecting patient confidentiality, managing consent, and ensuring data minimisation. Additionally, GDPR introduces the “right to erasure,” which conflicts with permanent storage models [10], [11].

These issues highlight the need for stronger security measures that go beyond traditional database logging and policy enforcement.

2.3 Blockchain fundamentals

Blockchain technology first appeared with Bitcoin in 2008 as a decentralised ledger system. Each block contains a cryptographic hash of the previous block, making tampering very difficult, hence immutable. Consensus protocols like Proof-of-Work and Proof-of-Stake ensure that blocks are added in a specific order among multiple participants [7].

Key properties include the following:

- Tamper evidence: Any change to historical records leads to breaking the hash chain.
- Chronological ordering: Blocks are linked one after another, providing a reliable timeline.
- Distributed trust: In multi-node settings, consensus protocols stop from making changes alone.
- Verifiability: Someone with access to the ledger can check the validity of blocks using hash and consensus rules.

Although blockchain was first used for cryptocurrencies, its features have drawn attention in areas that need strong proof of origin, such as supply chain, finance, and healthcare.

2.4 Blockchain in Healthcare

The healthcare field gains specific advantages from blockchain's tamper proof quality.

Medical records are sensitive and have long life spans. They are also subject to legal scrutiny.

Blockchain can:

- Provide integrity by making sure clinical data cannot be changed retroactively without being noticed [12].
- Enable auditability through clear logs of data access and updates [12].
- Support patient centred control by connecting records with flexible consent systems[3].
- Improve interoperability using shared standards based on blockchain digests [8].

Prototypes like MedRec, MeDShare, and OmniPHR show how blockchain can manage permissions, share data among institutions, and track access histories. However, most use a hybrid approach, keeping clinical data off the blockchain and only linking hashes or metadata on chain. This may reduce the size of the ledger, but it also creates reliance on traditional storage systems [13].

2.5 Regulatory and Ethical Considerations

The use of blockchain in healthcare raises important regulatory questions. HIPAA in the US and GDPR in the EU both require strong privacy protections and limit unauthorised data sharing. GDPR presents specific challenges, such as the right to erasure, which conflicts with the unchangeable nature of blockchain [11], [14]. Creating blockchain-based EMRs requires making trade-offs between permanence and compliance [14]. Proposed solutions include off-chain storage, hashed references, and privacy-preserving commitments to meet these needs.

Another ethical issue is finding a balance between patient autonomy and data security. While unchangeable records enhance accountability, patients need to maintain some control over their consent, including the ability to revoke access or limit sharing with third parties [15].

2.6 Positioning for this study

This project sits within a broader context by using a lightweight, integrity-first design. Unlike most previous methods that only save digests on-chain, the prototype stores complete EMR content in the ledger. This choice focuses on verifiable integrity and protection against insider threats rather than availability or sharing between institutions. The goal is to assess whether a single-node, application-level blockchain can provide valuable tamper evidence while still being usable in daily clinical workflows.

3 Literature Review

This chapter examines the application of blockchain in healthcare data. It compares tamper-evident integrity to policy-based logging. It outlines consent and access models and discusses whether to store data on-chain or off-chain. We conclude by presenting this prototype: a full EMR on-chain, single-node, PoW, serving as a local deterrent, lab OTP challenge, and with limited third-party reads

3.1 Blockchain for healthcare data

Prior EMR reports often use a mixed storage model. Clinical content is stored off-chain, while hashes and metadata are kept on-chain. These designs usually depend on permissioned ledgers with smart contracts for coordination and auditing. These methods decrease ledger size and make privacy management easier, all while ensuring provenance among stakeholders [7], [8], [10], [12], [14]. Key directions include interoperable PHR architectures (OmniPHR), access- and audit-centric ledgers (MedRec), and inter-provider sharing frameworks (MeDShare). It also promotes clinical-trial transparency through smart contracts [13]. Other studies focus on secure storage and sharing of medical information, as well as efficient blockchain-based exchange in healthcare ecosystems [4].

The existing system stores EMR content off-chain, creating a clear access trail [12]. MeDShare enables trustless sharing between providers by storing artefacts off-chain and accountability metadata on-chain [5]. OmniPHR suggests a distributed PHR architecture for better interoperability [8]. Early prototypes showed how blockchain can support secure EMR sharing between institutions [9]. Gordon and Catalini pointed out that patient-driven

interoperability is an important factor for adoption [11]. Meanwhile, MedBlock emphasises secure and efficient exchanges with on-chain proofs [3]. Other studies explore blockchain-assisted EMR exchange, decentralised patient-data management, and privacy-preserving contracts [13].

While the existing work also prioritises scalability and interoperability across multiple providers, they also use permissioned or consortium blockchains and keep the EMR off-chain. Our prototype keeps the EMR on-chain, prioritising tamper-evident integrity over scalability.

3.2 Integrity and auditability

Traditional audit logs, which include triggers and append-only tables, are enforced by policy and can be changed by someone with high access within the same trust domain as the data [1]. In contrast, hash-linked ledgers make it easy to spot past changes. Any edit alters a block's content digest or breaks the link to its previous block. Detection happens manually via structural checks (link equality, 00000 PoW prefix, and per-run timestamp monotonicity). Proof-of-Work raises the effort to change history while keeping verification cheap [3], [7], [16]. In summary, the order in which blocks are added determines provenance and timestamps, and only the temporal context [5], [12].

Instead of using the existing automated integrity checks, we deliberately include a manual verification routine to keep the lightweight and auditable protocol and lower operational complexity, but still deter tampering in a local deployment.

3.3 Consent and access models

The literature often discusses access to medical data through patient-centred consent and detailed permission settings. Early blockchain personal health record (PHR) and electronic medical record (EMR) systems use on-chain permission management or smart contract rules to control which users can read or write specific items. They often separate identity from access decisions to reduce the risk of sharing personally identifiable information (PII) [5], [8], [12].

Consent is often dynamic. Patients can grant, refine, or revoke their sharing preferences over time. Some models show this with clear contract logic or records of consent. Proposals for dynamic consent focus on revocation and the ability to audit as policies change. A broader survey shows that distributed ledgers can support fine-grained, patient-centred control of

biomedical data [11]. Additionally, privacy-preserving frameworks stress the need to minimise data and reduce access to metadata. They suggest avoiding the storage of sensitive access trails, like who queried what and when, on a public or consortium ledger. Instead, these details should remain off-chain or only hashed to limit the risks of linking and making inferences [5], [8], [14].

Our prototype uses fixed role-based permissions with OTP verification for labs. It trades dynamic consent flexibility to reduce PII exposure and avoid on-chain metadata leakage.

3.4 Storage placement trade-offs

Off-chain EMR (hybrid designs):

Pros: smaller ledgers, easier redaction, use of mature DB tooling and indexes.

Cons: the integrity of content depends on the off-chain store, verification requires fetching the artefact and matching it to on-chain digests [5], [8], [10], [12]

Full on-chain EMR (integrity-first designs):

Pros: self-contained verification via structural checks and link checks (link equality, 00000 PoW prefix, and per-run timestamp monotonicity).

Cons: larger ledgers; applications must provide efficient read paths; privacy must be engineered carefully (store only what is necessary; keep reads/auth off-chain); availability remains out of scope in single-node/prototype settings [1], [3], [7], [16].

Prior work typically adopts hybrid storage designs (EMR off-chain, digests on-chain), while our prototype takes a full on-chain approach. A detailed comparison of storage trade-offs is provided in Appendix C (Table 5) [2], [3], [5], [7], [8], [12], [16].

3.5 Gap and positioning

Much of the literature focuses on enabling multi-party availability and exchange between institutions through permissioned consensus and off-chain content storage. There is comparatively less focus on designs that prioritise low dependency and high integrity [1], [5], [7], [8], [10], [12]. These designs aim to provide cryptographic evidence against tampering while keeping clinician workflows straightforward. Patient-experience analytics further support the need for reliable data pipelines that maintain the origin of the data. This is crucial since downstream analytics rely on the integrity of upstream clinical records [17]. Patient-experience analytics also highlight the importance of provenance. Downstream analyses depend on unaltered clinical records [6], [17]. While previous studies focus on multi-party

exchange and patient interoperability, these designs often miss integrity-first approaches [9], [11], [15].

Unlike MedRec and MeDShare, our prototype focuses on integrity-first evidence at the point of care. It ensures tamper resistance even in single-node, low-trust environments.

4 Problem Analysis

Electronic Medical Record (EMR) systems are essential in today's healthcare as they support diagnosis, continuity of care, and legal accountability. However, current implementations depend heavily on centralised databases and the trust placed in administrators, which leads to significant vulnerabilities [1], [4].

4.1 Insider threats

Privileged insiders, such as administrators or database operators, can change records without being detected. They can disable or rewrite audit logs. This allows for retroactive edits, replays, or back-dating of clinical data, all of which undermine medical integrity and patient trust [7]. Such manipulations compromise the clinical decision-making process and threaten legal compliance. The authenticity and timeline of records are crucial for resolving disputes [5], [7], [12], [16].

4.2 Weaknesses of Hybrid Blockchain-Database Models

An issue comes from the hybrid blockchain-database models found in much of the literature. In these systems, only hashes or metadata are stored on-chain while the main clinical content stays off-chain. Although this reduces ledger size and simplifies interoperability, it also creates a dependency on the off-chain database. Insiders can still alter the underlying EMR data and recompute digests to hide their actions. This setup weakens the tamper-evidence that blockchain is supposed to provide [5], [8], [10], [16].

4.3 Limitations of Traditional Audit and Cryptographic Methods

Traditional audit methods, such as triggers and append-only tables, stay within the same trusted environment as the data. This makes them vulnerable to insider abuse. While cryptographic signatures and timestamps assist with authorship, they still permit systematic rewriting of history [1].

4.4 Usability vs Security Challenges

Public and consortium blockchains make data available, but they are expensive and slow. For small clinics, these options are not workable. This shows the need for simple, integrity-focused solutions [10].

4.5 Missing Role Diversity

Access control and role diversity in clinical settings are often missed. Most prototypes only look at doctor and patient interactions. However, real workflows include laboratories that need to upload verified results and third parties like caretakers who may need limited read-only access [3], [8].

4.6 Need for Integrity-First EMRs

Addressing these issues requires an EMR system that is tamper-evident, can be verified using cryptography, and resists manipulations made after the fact. It also needs to be lightweight enough for use in a single-node clinical setting. This project meets that need by proposing a blockchain-based EMR prototype. It ensures integrity from start to finish while supporting practical healthcare workflows tailored to specific roles [3], [10].

5 Project Specification

This chapter explains the goals, scope, and requirements of the prototype EMR system. It constructs user needs based on their roles, how the platform behaves, its non-functional qualities, and the specific constraints and assumptions.

5.1 Software methodology

We used an Agile, Scrum-inspired method tailored for a solo developer project. While Scrum traditionally requires a full team, the rituals were simplified for a single developer context:

- **Sprint Planning and Backlog:** Tasks were tracked in Jira as personal user stories. A snapshot of the backlog view, captured mid-development, is included in Appendix F (Figure 12) as evidence of sprint management.
- **Daily Standups turned into Personal Checklists:** Instead of team standups, daily checklists and progress notes were maintained.
- **Sprint Reviews and Retrospectives became Supervisor Feedback:** End-of-sprint outcomes were reviewed in meetings with my supervisor, who offered feedback and guidance.

Sprint focus (3 cycles):

- Sprint 1, Blockchain core: block structure, SHA-256 hashing, PoW (difficulty 5), JSON persistence.
- Sprint 2, Server & access: TCP socket protocol, role permissions (Doctor/Patient/Lab Technicians/Third-party), email OTP flow (transient and not stored), DB tables for users/appointments.
- Sprint 3, UI & hardening: Swing screens, backup/restore routine, manual link/PoW/timestamp verification procedure.

This lightweight Scrum adaptation allowed for quick iterations while keeping deliverables clear and easy to review.

5.2 Requirement gathering

See Appendix D for the Interviews which were conducted with the doctor, patient, lab technician, and third-party (caretaker/friend of the patient) to gather the requirements for the dashboard, which is being added along with personas (Table 6-9).

5.3 System requirements

- Integrity (primary): Provide cryptographic tamper evidence for all clinical writes using SHA-256 hash links and Proof-of-Work (difficulty = 5). Integrity was a priority because tamper-evident EMRs are essential for healthcare audits and malpractice disputes.
- Practicality: Keep the system light on dependencies for small clinics: single-node server, Java Swing clients, TCP sockets.
- Data placement: Store all clinical EMR content on-chain (JSON ledger). Keep user and appointment metadata in MySQL.
- Verifiability: Allow manual end-to-end verification after backup and restore using hash links, PoW prefix, and timestamp monotonicity for each mini-chain.
- Role separation: Establish clear, least-privilege flows: Doctor (accept/reject appointment, write report), Lab (write with OTP), Patient (book appointment/view report), Third-party (read-only using Appointment ID and DOB).

5.4 Functional requirements

5.4.1 Ledger and storage

- FR-1 Append-only writes: Each clinical write forms a block {aid, problem, test, treport, report, timestamp, previoushash, hash}, linked by previoushash.
Acceptance: Verified when each block's previoushash matches the hash of the block before it.
- FR-2 PoW mining: Before appending, compute a hash with five leading hex zeros (difficulty = 5).
Acceptance: Stored hash begins with 00000.
- FR-3 JSON persistence: The ledger is a JSON file. It loads at startup, appends and clears on each write.
Acceptance: After a restart, the chain reflects the last committed head.
- FR-4 Mini-chains on restart: Each server restart begins a new mini-chain with a fresh genesis. Ordering and manual verification follow each mini-chain.
Acceptance: The first block's previoushash equals the configured genesis marker ("0").
- FR-5 Database has metadata only: MySQL stores user management and appointments, including Appointment ID, Appointment status, and User details
Acceptance: There is no EMR block content and no OTP values in the database.

5.4.2 Access and flows

- FR-6 Doctor writes: A doctor can create or update EMR entries linked to an Appointment ID.
Acceptance: A successful write creates a new on-chain block that appears in the patient's history.
- FR-7 Lab technician writes with OTP: Upload requires an OTP challenge sent to the lab's registered email. The system validates the OTP in memory or session, but does not save it.
Acceptance: The upload is accepted only after receiving a valid OTP within the allowed time frame.
- FR-8 Patient read: The patient can view their own EMR in read-only mode.

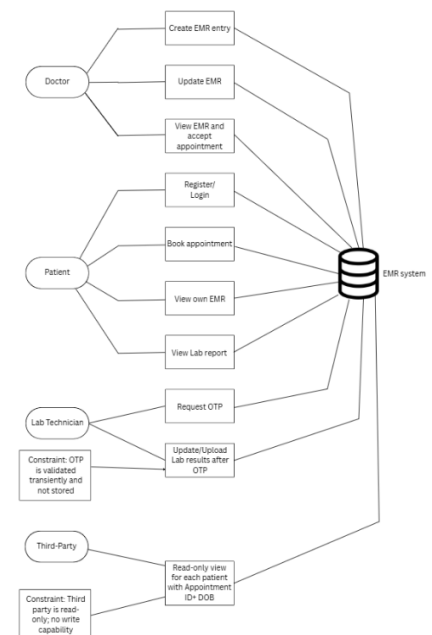


Figure 1: Use-case diagram of 4 modules

Acceptance: The user interface displays blocks for the patient arranged in canonical (append) order with the correct details.

- FR-9 Third-party read: A third party can view read-only EMR for one patient when the Appointment ID and DOB match.

Acceptance: Access is granted only with an exact match, where write controls are not given.

- FR-10 On-chain scope: The ledger records write from doctor updates and lab uploads. Reads and authentication remain off-chain.

Acceptance: No read or login events appear on the chain.

5.4.3 Verification and recovery

- FR-11 Manual verification routine: Provide a procedure to:

(a) split into mini-chains, (b) check linking with previous hash, (c) validate PoW prefix (00000), (d) ensure non-decreasing timestamps for each mini-chain.

Acceptance: The routine reports the last verified head. Any broken link, prefix, or timestamp flags the problematic block.

- FR-12 Backup/restore: After restoring the JSON ledger and MySQL schema, run FR-11 to confirm integrity from start to finish.

Acceptance: A clean verification report means a valid restore, and any discrepancies are noted.

Note: The mining nonce is not saved; full target re-computation will be addressed in the future.

5.5 Non-functional requirements (NFRs)

- NFR-1 Integrity: Tamper evidence comes from SHA-256 hash links and proof of work. Detection occurs through manual verification with link, prefix, and timestamp checks.
- NFR-2 Privacy: Stores important clinical content on-chain. Do not record read or access events on the ledger. Keep authentication artefacts off-chain and minimise exposed identifiers.
- NFR-3 Usability: Provide a clear Swing user interface. Include informative messages for invalid OTP and invalid ID, or date of birth.

- NFR-4 Recoverability: Support backup and restore of the JSON ledger and MySQL/HeidiSQL schema.
- NFR-5 Durability: When appending, clear the new block to disk before confirming the write.
- NFR-6 Maintainability: Use a modular Java server with separate components for the ledger, mining, database access, and socket protocol. Make the proof of work difficulty adjustable.
- NFR-7 Performance: Block appending, which includes mining and persistence, remains interactive when Proof-of-Work with a difficulty of 5 (see FR-2).
- NFR-8 Interoperability: Support export and import through JSON backup.

5.5.1 Accessibility (non-functional)

The application is built for accessibility. It can be used by patients, doctors, lab technicians, and administrators of various ages and abilities. Healthcare systems need to serve a broad audience, so the design meets the WCAG 2.1 Level AA standards and takes age-specific accessibility into account.

- WCAG Compliance: Logins, menus, and blockchain actions are operable via keyboard. They have clear labels and high contrast text for better readability. Error messages use plain language.
- Child-Friendly: The system features simple story-like flows (Book Doctor, Visit Lab, See Report), large buttons, and dropdowns to reduce typing.
- Senior-Friendly: It uses large fonts, provides logical linear navigation, has confirmation prompts for critical actions, and does not enforce strict timeouts.
- Robustness: The system works on Java-supported devices and at different zoom levels.

The accessibility requirements are detailed in Appendix E (Tables 10–13).

Age-specific accessibility guidelines are provided in Tables 14 and 15.

5.6 Constraints and assumptions

These limits are intentional design choices for a prototype in a controlled LAN clinic setup.

- Single-node scope: There is no replication or multi-node consensus. Availability is not included as this prototype runs on a single server.

- **Trusted network:** The prototype runs on a trusted local area network over TCP sockets. TLS is not included.
- **Time source:** The system clock timestamps enforce monotonicity for each mini-chain.
- **OTP delivery:** Email must be accessible. OTPs are generated and validated temporarily. OTPs are never stored.
- **Filesystem:** The JSON ledger depends on operating system permissions and backups. Physical immutability is out of scope for this prototype.
- **Nonce persistence:** The mining nonce is not stored. This limits full proof of work re-computation, as detailed in future work.

5.7 Roles and permissions matrix

This matrix follows the principle of least privilege. Each actor has only the minimum required capabilities.

A= Allowed, R= Read-only, D= Denial, - = Not applicable

Operation / Role	Doctor	Lab technician	Patient	Third-party
Register / Login	A	A	A	-
Book appointment	-	-	A	-
View own EMR	-	-	A	-
View patient EMR	A	-	A	R
Create EMR entry	A	-	-	-
Update EMR entry	A	-	-	-
Upload lab result (on chain)	-	A	-	-
On-chain access logs (read)	D	D	D	D

Table 1: Roles and permission matrix

5.8 Out-of-scope

These exclusions set the project boundaries and show intentional simplifications:

- **Access-history and logs:** Provide a user-friendly access-history view. There is no patient-facing "who accessed my record" dashboard.
- **Transport encryption (TLS):** TLS is not included since the prototype assumes a trusted local area network

- Mobile and notifications: In this prototype, the target is only for the Java Swing desktop client. Android/iOS apps are not included.
- Persisting mining nonce and adding automatic full-chain verification on startup

5.9 Project Management

The project used GitLab for version control and task tracking. All source code, commits, and issue logs were stored in a private GitLab repository (see Appendix A). This kept track of development activities and allowed for team collaboration on version management.

To keep track of progress, a Gantt chart was created at the start of the project and updated regularly. It showed task dependencies, milestones, and deadlines, which helped keep the project on schedule. A complete version of the Gantt chart is in Appendix F (Figure 11).

6 Solution Design

6.1 Architecture overview

The prototype system uses a client-server architecture, as shown in Figure 2. Four specific Java Swing clients are Doctor, Patient, Lab Technician, and Third-party, who communicate with a Java socket server using a simple TCP request and response protocol. The server has two separate storage components: a blockchain ledger for clinical records and a relational database for user and appointment metadata.

- Blockchain ledger implemented as a JSON file: Full EMR content stored as SHA-256 hash-linked blocks, mined with Proof-of-Work at difficulty 5, which means five leading hexadecimal zeros.
- MySQL: stores user accounts and appointment metadata, including Appointment ID, role, and login. No OTPs and third-party access are stored.

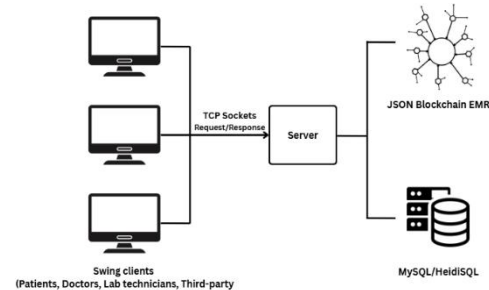


Figure 2: End-to-end data path and storage split

6.2 Data placement and models

The prototype maintains a clear separation between on-chain and off-chain data. The blockchain ledger records only clinical write events. Each block stores the fields {aid, problem, test, treport, report, timestamp, previoushash, hash}. This setup guarantees that clinical content can only be appended. The relational database (MySQL) handles user accounts and appointments. It stores credentials, roles, Appointment IDs, and patient

connections, along with a small number of operational metadata. It's important to note that OTP values are never saved. They are generated and sent to the lab's registered email, briefly validated in memory or during the session, and then discarded.

This meets FR-1 (append-only ledger) and FR-7 (secure OTP flow) because OTPs are validated only in memory and are never stored.

6.3 Block structure and mining

Field	Type	Description
aid	int/String	Appointment ID linking the write to a patient
problem	String	Presenting complaint
test	String	Tests needed to be performed
treport	String	Test report
report	String	Clinician diagnosis
timestamp	long	System clock when mined
previoushash	hex string	Hash of previous block (if genesis, it is marked as '0')
hash	hex string	Stored PoW hash for the block (starting with 00000)

Table 2: Block fields and descriptions

Blocks are linked to previoushash and mined to difficulty 5 (00000 prefix). Ordering is appended sequence, and timestamps are additional.

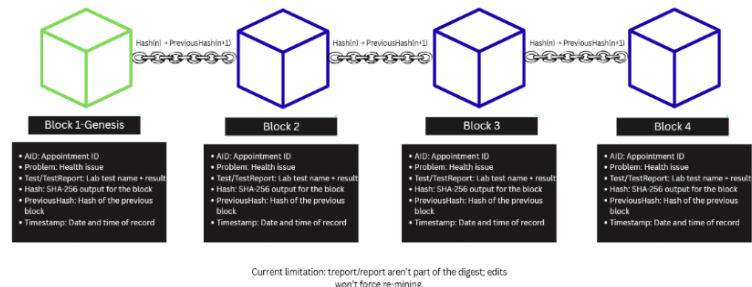


Figure 3: Block linkage and Proof-of-Work structure

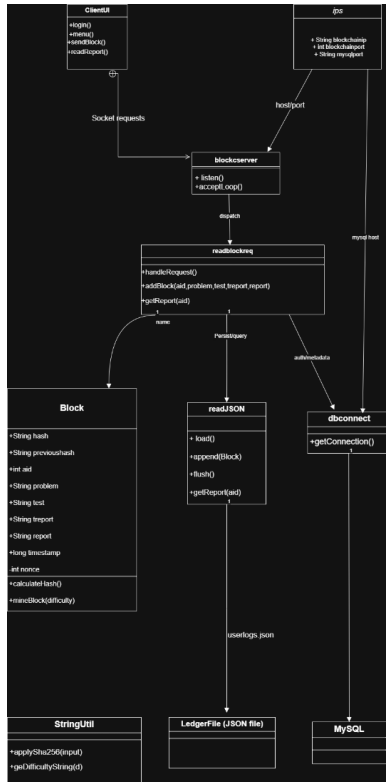
6.4 Socket protocol

A complete message contract, including request and response payloads, JSON examples, and error codes, together with a detailed sequence diagram of the Doctor write path, which covers the ADDBLOCK request, mining, and acknowledgement, is in Appendix G (Figure 16)

- Server details: Listens on ServerSocket (3000) and handles each connection by parsing the request code with arguments per connection.
- Mining: Only when adding a new block (ADDBLOCK), mineBlock (5) ensures the hash starts with 00000 and advances the head.

- In the current implementation, hash does not include treport/report (hash is over previousHash + timeStamp + nonce + aid + problem + test). Therefore, UPDATETESTREPORT / UPDATEDOCTORREPORT do not trigger re-mine and does not break the chain.

6.5 Access control model



- Doctor: Create/update EMR entries (on-chain writes linked to Appointment ID).
- Lab technician: Permitted on-chain write only after OTP challenge is delivered to the lab's registered email and login is completed when the entered OTP is correct.
- Patient: Read their own EMR and book appointments (DB).
- Third-party: Read-only view for a patient when the Appointment ID + DOB are correct, and are not allowed to write.

Figure 4: Class diagram of all implemented modules

6.6 Rationales

6.6.1 Customised blockchain over a database only

A common database storage only is not preferred because:

- A privileged insider, such as a DBA or root user, or a compromised service, can modify clinical data and bypass the database's own audit mechanisms. Controls, including temporal tables, CDC/CT, binary/ledger logs, triggers, and stored procedures, exist within the same trust boundary as the data. An administrator can disable, purge, or rewrite these controls.

Design choice: Used MySQL for user management and appointments, along with general querying. Add a lightweight, application-embedded blockchain for clinical writes. Each write is committed as a SHA-256 hash-linked block mined to a difficulty of 5, meaning five

leading hex zeros. Manual verification is required. Simply scan the ledger file once and check the previous hash links, PoW prefix, and non-decreasing timestamps for each mini-chain. A new mini-chain starts after each server restart.

Other approaches considered but rejected for this prototype, because:

- Public blockchains (Bitcoin, Ethereum):
 - Bitcoin processes about 7 transactions per second, with high fees and energy use.
 - Ethereum gas fees change unpredictably, making regular EMR writes too expensive.
- Permissioned Blockchains (Hyperledger Fabric, R3 Corda):
 - Our prototype serves only one clinic or institution, so a multi-party agreement is not needed.
 - Network partitioning and Byzantine fault tolerance deal with availability issues that fall outside our scope.
 - Consensus protocols add delay even when just one node is involved in writing.
 - Endorsement policies need multiple peer validations for simple EMR entries.
- Consortium blockchains:
 - Platform updates may disrupt existing integrations or change consensus rules.

Advantages of using the customised blockchain:

- Any retroactive change to hashed inputs alters the digest and breaks the link to the next block. Verification identifies the first failing block.
- The order in which items are added is canonical. Timestamps offer context but do not establish authority. In this prototype, the order is per mini-chain. Cross-run stitching is future work.
- Verification needs only the ledger JSON file and SHA-256. There is no need for database credentials or audit configuration.
- Hash-linked blocks focus on tamper-evident provenance without added features.
- They do not require network consensus, mining pools, or external validation nodes.
- There is no need for block explorers or third-party verification services.
- You do not need blockchain expertise or special hardware.
- The restart policy, or fresh genesis, works well with episodic clinical operations.

6.6.2 Why Proof-of-Work

We need tamper-evident history on a single server with minimal dependencies. The main risk comes from a trusted insider who might edit old records and try to compute a fake chain.

Why not the common alternatives:

- Plain hash-chain: It provides linkage but no cost. A forged tail can be quickly recomputed.
- Signatures-only: They prove authorship but not the order of appending. If the key is compromised, the assurance is lost.
- DB temporal/CDC/audit + SIEM: These are useful for operations but remain within the same admin domain. A DBA can disable, purge, or alter both the data and logs.
- RAFT/PBFT (permissioned consensus): These solve availability and agreement across nodes but are operationally heavy and not suitable for a single-node integrity prototype.
- Trusted timestamping/WORM: These can assist with retention and time anchoring, but do not create a re-computation cost for rewriting history at the application level.

Advantages of using PoW:

- Rewrite cost: Each block must meet a difficulty target of five leading hex zeros at 5. If block k is changed, the attacker must re-mine blocks k and all subsequent blocks. With fixed difficulty, the cost increases with the number of subsequent blocks.
- Simple and manual verification: Walks through the ledger, checks previous hash links and the 00000 prefix, and verifies monotonic timestamps for each mini-chain. No extra infrastructure or database policy is required.
- Right fit for single-node: PoW adds a practical choice without the need for permissioned clusters, certificates, or external services. It suits a single-node prototype.

The difficulty parameter is fixed at 5 for the following reasons:

- If the difficulty is less than 5, rewriting history becomes too cheap. An attacker could re-mine a fake tail in milliseconds. Setting the difficulty to 5 creates enough computational cost to make rewriting noticeably slower, while still allowing quick additions in less than a few seconds on a modern laptop or PC.

- In Bitcoin or Ethereum, the difficulty is high because thousands of miners compete. However, in our single-node prototype, we do not want mining to take minutes. That would affect usability for doctors and labs entering EMRs.
- Difficulty 5 is the right balance:
 - Appends are fast enough for responsive clinical use and ensure good responsiveness.
 - Rewrites require significant effort, so tampering is difficult.
 - It shows the security principle without needing special mining hardware (unlike Bitcoin mining).

Table 17 in Appendix H gives a detailed comparison of options. PoW with difficulty 5 was chosen as the best approach.

6.7 Verification procedure:

The ledger can be checked after a backup restore or whenever required by re-hashing each block. This process confirms the Proof-of-Work target, which requires five leading zeros, and checks the link to its parent block. Timestamps are verified to ensure they are in order, and any block that doesn't pass these tests is marked as invalid or orphaned. Subsequent writes continue from the last verified point, maintaining the sequence. Currently, this verification process is conducted manually. Future work will make this process automatic at startup, save the mining nonce, and isolate any invalid tails.

7 Implementation

7.1 Run and deploy

The system needs a Java SE environment (JDK 8 or 11), a MySQL server set up with the emrsharing schema, and some extra libraries (MySQL Connector/J, JSON-Simple, JavaMail, JCalendar). Start the server using blockchainServer (blockcserver), which opens a TCP socket on port 3000. After that, client interfaces (doctorlogin, patientlogin, lablogin, guestlogin) can be opened from NetBeans to carry out role-based tasks.

Deployment follows a standard flow.:

- When the doctor creates EMR blocks, the blocks are mined at difficulty 5 and added to the JSON ledger.
- The patient views records by appointment ID. The lab technician uploads report after OTP-based verification.

- Third-party users can only view records in read-only mode.
- Integrity is checked by looking at hash linkages in userlogs.json.

Deployment notes and troubleshooting are in Appendix I.

7.2 Technology stack

- Language and UI: Java (SDK 8+), Swing, Netbeans GUI Builder (drag-and-drop components)
UI built using Java Swing (drag-and-drop via Netbeans GUI builder). See Appendix I for the detailed component function mapping.
- Dependencies: MYSQL Connector/J 8.0.x; JavaMail (javax.mail 1.6.x)
- Transport: TCP sockets (Java ServerSocket/Socket, ObjectInputStream/ObjectOutputStream)
- Integrity: SHA-256 hashing (utility class), Proof-of-Work (difficulty = 5)
- Persistence: JSON ledger file for clinical writes
- MySQL (administered via HeidiSQL) for users and appointments
- Email (OTP): JavaMail (e.g., Gmail SMTP) via SendMailExample.java (OTP validated in memory, not persisted)
- Platform: Single server (Windows/Linux), trusted LAN

7.3 Server components

- blockcserver.java: Opens a ServerSocket on port 3000. It accepts client connections and reads a command String with typed arguments. The system then dispatches to handlers and sends back a status string, like "SUCCESS", "FOUND", "NOTFOUND", or an error, along with any return fields. It supports the following operations (see section 4.4): ADDBLOCK, GETMYREPORT, GETTESTREPORT, GETTEST, UPDATETESTREPORT, and UPDATEDOCTORREPORT.
- readblockreq.java: It is the per-connection request processor which parses the command String, calls the right ledger or database routine, and serialises the reply. It ensures append order by reading the current head and setting the previoushash for new blocks.
- readJSON.java: It is the ledger I/O which loads the JSON array at startup if it exists, appends new blocks, and flushes to disk. Upon restart, it begins a new mini-chain with a genesis marker, so verification occurs for each mini-chain.

7.4 Blockchain core

- block.java: Fields that are stored in the JSON are {aid, problem, test, treport, report, timestamp, previoushash, and hash}(linked by previoushash) mined to a 5-hex-zero prefix (FR-2). The mining loop increases a nonce, which is kept in memory, until the hash starts with five leading hex zeros (00000). When this is successful, the block is added. The nonce is not saved
- Hash input = prevHash+time+nonce+aid+problem+test whiletreport/report not hashed, so edits do not re-mine. This is noted in section 4.6 as a known limitation with the planned fix. It will include all changeable clinical fields in the digest and save the nonce to allow complete re-computation on startup.
- StringUtil.java: SHA-256 helper (applySha256), and convenience methods for difficulty checks (e.g., getDifficultyString(5)).

```
public void mineBlock(int difficulty) {
    String target = new String(new char[difficulty]).replace('\0', '0');
    while(!hash.substring(0, difficulty).equals(target)) {
        nonce++;
        hash = calculateHash();//Brute force mining process
    }
    blockserver.jTextArea1.append("Block Mined!!! : " + hash+"\n");
    readblockreq.previousHash=hash;//ensures block linkage
}
```

```
public static String applySha256(String input){
    try {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(input.getBytes("UTF-8"));
        StringBuffer hexString = new StringBuffer();
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString(0xff & hash[i]);
            if(hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        return hexString.toString();
    } catch(Exception e) {
        throw new RuntimeException(e);
    }
}
```

Figure 5: Blockchain core code

7.5 Clients

- Doctor (doctorlogin.java, doctorregister.java, doctormenu.java, sendblock.java)
Functionalities: Login/register; create new EMR block (on-chain) for an appointment (ADDBLOCK, FR-6/FR-10); update doctor narrative (UPDATEDOCTORREPORT); view combined test+report (GETTESTREPORT).
- Patient ((patientlogin.java, patientmenu.java)
Functionalities: Login; View own EMR by Appointment ID (GETMYREPORT, FR-8); book appointments (DB).
- Lab technician (lablogin.java, labmenu.java, SendMailExample.java)
Functionalities: Login using OTP. If successful, the user can upload test reports.
- Third-party (guestlogin.java, guestreport.java)
Functionalities: Authenticate with Appointment ID + DOB; read-only EMR view via GETMYREPORT. The UI lists the patient's closed appointments for selection and

shows problem, tests, lab report, and doctor report. No write operations can be performed

```
try
{
    Authenticator auth = new SMTPAuthenticator();
    Session session = Session.getInstance(props, auth);

    MimeMessage msg = new MimeMessage(session);
    msg.setText(message_mi);
    msg.setSubject(subject_to_be_given);
    msg.setFrom(new InternetAddress(sender_email));
    msg.addRecipient(RecipientType.TO, new InternetAddress(reciever_id));
    Transport.send(msg);
}
catch (Exception mex)
{
    mex.printStackTrace();
    reply="FAILED";
}

return reply;
```

Figure 6: OTP mail dispatch

7.6 Database access and schema

- Dbconnect.java: Centralises JDBC connection to MySQL using JDBC drivers and IP address defined in ips.java. The connection is used by login/registration screens and appointment flows.

```
Connection con=null;
try
{
    Class.forName("com.mysql.jdbc.Driver");
    con=DriverManager.getConnection("jdbc:mysql://"+ips.mysqlip+":3306/emrsharing", "root","root");
}
catch(Exception e)
{
    System.out.println(e);
    e.printStackTrace();
}
```

Figure 7: JDBC connection in Dbconnect.java.

```
public class ips {
    public static String blockchainip="localhost";
    public static int blockchainport=3000;

    public static String mysqlip="localhost";
}
```

Figure 8: Runtime endpoints in ips.java.

- Data stored in MySQL are Users- authentication and role metadata (user_id, username, gender, email), and Appointments- patient linkage and scheduling (aid, patient_id, doctor_id, date_time, status)

Connection path: Swing client → Socket server → Dbconnect.java (JDBC) → MySQL

7.7 Restart policy and persistence

- On each server restart, the system starts a new mini-chain (genesis previoushash = "0"). The ledger thus holds several runs over time. This makes recovery easier but resets global continuity with each restart.
- The data is stored in a plain JSON array of objects where each element holds a “block” object with the fields listed in section 4.3, along with the stored hash.
- On server startup, the JSON ledger loads to show past EMR records. New doctor/lab entries are mined, which means they are hashed with 5 zeros and added as blocks. Blocks that have "previoushash": "0" indicate new mini-chains after restarts. Other blocks connect to the hash of the previous block.
- Verification using manual link-and-prefix checks is done. Integrity is checked per mini-chain.

Advantages:

- If a system crashes in between mining, this does not risk a half-finished block to corrupt the entire global chain so that no medical data is lost.
- Clinics can archive old mini-chains every month or quarter. They should keep only the latest one active.



```
"block": {
  "problem": "ear pain, throat pain",
  "test": "ear scan",
  "previoushash": "0",
  "treport": "an obstacle is found",
  "report": "Dont rub your ears, consult tomorrow morning",
  "aid": 34,
  "hash": "00000f2a9b045598655d8eb5b95de3402299420b575943f799614e1af06a4784",
  "timestamp": 1755421583719
}

"block": {
  "problem": "red spots on hands and feet",
  "test": "blood test",
  "previoushash": "00000f2a9b045598655d8eb5b95de3402299420b575943f799614e1af06a4784",
  "treport": "infection found",
  "report": "foot and mouth disease detected",
  "aid": 35,
  "hash": "00000b36c51f155919502dd1368d5a811ac252112a3d956f31bf185a52f2bfcf",
  "timestamp": 1755422569788
}
```

Figure 9: Blockchain block objects in JSON showing tamper-evident links via *previousHash* and *hash* fields.

7.8 Error handling and logging

- Sockets & I/O ensure each request is wrapped in a try/catch block. Network failures return a simple error status to the client and log `Exception.getMessage()` with a stack trace in the console.
- Input validation is basic null and empty checks on client forms. The server trusts types because the protocol is typed through object streams.
- Multiple clients are served at the same time. We fixed UI responsiveness issues using multithreading and handling runtime exceptions.
- Console logs using `System.out.println` capture major events.

8 Evaluation and results

Reporting approach: This prototype focuses on cryptographic integrity instead of throughput. We report hardware-agnostic properties, such as expected Proof-of-Work trials at difficulty = 5 and $O(N)$ verification. We back these up with ledger evidence, including block count, link/prefix checks, and bytes per block. We purposely do not include wall-clock timings, as these depend on the machine and do not affect the design guarantees. A few seconds per block was observed in practice.

8.1 Objectives and approach

We check if the prototype meets its goals:

- Tamper-evident integrity for clinical writes (hash-links + Proof-of-Work with difficulty=5)
- Correct role workflows over sockets (Doctor/Patient/Lab/Third-party)

- Interactive performance while mining at difficulty 5.

We use risk-based testing where we test the highest-impact paths first, like ledger integrity, OTP for lab writes, and third-party read scope. Then we cover negative cases and regression.

8.2 Testbed

- Hardware/OS: Windows 11 (64-bit), 8 GB RAM.
- Runtime: Java SE 8 (tested; also runs on 11/17 LTS), Swing clients
- Server: single process; TCP sockets (ServerSocket/Socket)
- Integrity: SHA-256 + PoW d=5 (five leading hex zeros)
- Persistence: JSON ledger (clinical writes); MySQL/HeidiSQL (users, appointments); OTP via SMTP (transient, not stored)

8.2.1 How we manually verify

1. Open the JSON ledger (userlogs.json) and find consecutive blocks i-1 and i.
2. Link check: confirm that the previous hash of block i equals the hash of block i-1.
3. PoW prefix check: confirm that each hash starts with 00000 (difficulty 5).
4. Timestamp sanity: within a single run or mini-chain (genesis has previous hash “0”), timestamps should not decrease.
5. If any check fails, note the first failing index. Continue from the last valid head or start a new mini-chain when restarting.

This corresponds to NFR-1 (tamper-evidence).

To further validate these checks, we manually introduced tampering in the ledger file (userlogs.json) by editing hashes, breaking links, or replaying blocks. The results of these experiments, which included link failures, prefix mismatches, and timestamp anomalies, confirmed that the system is tamper-evident, except where certain fields are not hashed.

A complete set of tampering experiments and observations is provided in Appendix K.

8.3 Risk-based testing

This is what we test first:

Risk	Why risky	What we verify	Pass condition
Clinical writes on-chain	Integrity	Mining (d=5) + link/00000 checks	New block mined; manual check passes

Lab upload+OTP	External email dependency	Wrong/expired/replayed OTP denied; correct accepted	Negatives denied; positive writes on-chain; OTP not stored
Third-party read scope	Privacy	Appointment-ID + DOB exact match; read-only; write is rejected	Reads allowed only on exact match; writes rejected
Backup/restore	May introduce corruption	After restoration, the manual link-and-prefix checks are run	Pass or first failing index reported; next append from the last verified head
Performance with difficulty set to 5	Usability	Analytical model (PoW expected trials = 16^5); verification $O(N)$; storage linear	Model holds; ledger evidence matches linear trends

Table 3: Risk Assessment and Validation Strategy

8.4 Functional correctness

Functional testing validated core workflows across Doctor, Patient, Lab, and Third-party roles. We checked that the on-chain blocks matched the appointment data in MySQL. This ensured consistency between the blockchain ledger and the database records.

The complete set of 16 scenarios, including edge cases and error handling, is provided in Appendix J, supplementing the 4 shown in Table 4.

Sl.no	Scenario	Pre-conditions	Steps	Expected
1	Doctor creates EMR block	Doctor logged in; valid aid	Login → enter aid, problem, test, treport/report → submit	Server mines ($d=5$); new block visible in recent for aid
2	Lab upload with OTP	Lab logged in; registered email	Request OTP → receive email → enter OTP → upload treport	On success, write stored on-chain; OTP not persisted
3	Third-party read-only access	Appointment exists	Enter Appointment-ID + DOB → view	Read-only EMR shown; any write op rejected

4	Concurrent actions contention	Two clients active	Doctor ADBLOCK while Patient views	Both complete; no deadlock; consistent view
---	-------------------------------------	-----------------------	---------------------------------------	---------------------------------------------------

Table 4: Some Representative Functional Correctness Test Cases

8.5 Link-and-prefix Integrity tests (manual checks)

Goal: To demonstrate what the prototype can detect without an in-app verifier.

Method: Work directly on the JSON ledger. Then perform the manual link, prefix, and timestamp checks.

Takeaway: The prototype shows link-and-prefix tamper evidence through inspection. Silent edits to payload fields that aren't hashed or to hashed fields without re-computation won't be detected without a complete verifier. This is noted and set aside for future work.

8.6 Backup/restore integrity

- After restoring the JSON ledger and database, we carry out manual link and prefix checks with no in-app verifier. (1) Each previoushash[i] = hash[i-1]. (2) Each hash starts with 00000 (difficulty = 5). (3) Timestamps are non-decreasing within each mini-chain.
- The integrity is confirmed if all checks pass. If there is any corruption, the first broken link or invalid prefix is visible through inspection. Valid appends continue from the last verified head. A new genesis starts on restart, so verification and ordering follow the mini-chain.

8.7 Performance (PoW and I/O)

Goal: To show that appends stay interactive at difficulty=5, verification is linear with number of blocks, and storage grows linearly with event count.

8.7.1 PoW Cost Model

With a target of five leading hex zeros, the success probability per hash is $p = 16^{-5} = 1/1,048,576$. Therefore, the expected number of trials $E[\text{trials}]$ is about 1,048,576. In practice, at prototype scale, this means interactive mining times of a few seconds on standard hardware (see Section 8.7.4).

8.7.2 Verification Complexity

Verification is a single linear pass across the ledger ($O(N)$).

- Link check: `previoushash[i] == hash[i-1]` for non-genesis blocks.
- PoW prefix check: each stored hash begins with 00000.
- Timestamp monotonicity: timestamps do not decrease within each mini-chain. A new chain starts when `previoushash` equals "0".

Since the nonce is not saved, verification is limited to checking the structure and prefixes, not the complete proof-of-work re-computation. Observed ledgers have multiple mini-chains because of server restarts, but verification stays linear within each run.

8.7.3 Storage characteristics

Storage increases steadily with the number of clinical writes.

- Each block adds a fixed amount of overhead, including aid, timestamp, previous hash, and hash. It also includes variable clinical content, such as problem, test, treport, and report.
- As a result, the ledger size grows in a straight line with the number of events.
- Formatting choices, like pretty-printed versus minified JSON, may have a small impact on total file size. However, they do not change the linear growth pattern.

Takeaway: The storage overhead remains stable for each block. This means scalability can be projected reliably based on the expected volume of clinical activity.

8.7.4 Practical validation (qualitative)

Prototype testing confirmed the following:

- Responsiveness (observed): Prototype testing with multiple clients (~ 20 test runs, 4 roles concurrently), including Doctor, Patient, Lab, and Third-party, confirmed responsiveness on a small scale. All submitted blocks were mined and added without any noticeable delay or error.
- Integrity checks: Each block in the ledger met the conditions for linkage (`previoushash` = prior hash), Proof-of-Work prefix ("00000"), and timestamp order within each mini-chain.
- Storage behaviour: The growth of storage increased in a linear pattern that matched the measured block sizes, aligning with the expected $O(n)$ growth as events increased.
- Concurrency handling: Role-based operations, such as Doctor writes, Lab OTP challenges, and Patient/Third-party reads, ran correctly without conflicts or corruption in the JSON ledger or MySQL metadata.

- Validation scope: Observed responsiveness on standard hardware was a few seconds per block. This agrees with the analytical model. However, formal stress testing will be done in the future.

8.8 Comparative positioning

What others do: Many healthcare uses blockchain systems and keep EMR off-chain, and store only hashes and user management on a permissioned ledger. They prioritise availability for multiple parties. Focus on patient-doctor interactions only [5], [8], [12], [15].

This prototype: Full EMR is on-chain, using a single node and Proof-of-Work(difficulty=5) as local deterrence. It focuses on integrity without cluster dependencies. Furthermore, it allows lab technicians to upload using OTPs and provides read-only access for third parties, verified through patient details like date of birth and appointment ID. To our knowledge, no current system combines these multiple workflows with on-chain integrity guarantees.

Takeaway: It is simpler to deploy, but availability and privacy controls must be designed separately.

8.9 Threats to validity

The reported performance results come from a single-node prototype running with PoW difficulty set at 5. Mining latency and throughput will differ across various hardware platforms. The “few seconds per block” was seen on test hardware (Windows 11, 8 GB RAM; see Section 8.2).

Another limitation is the lack of nonce storage in each block. Without the nonce, the system cannot completely recompute historical hashes to verify Proof-of-Work. As a result, current verification is limited to structural checks:

- Hash linkage ($\text{previoushash}[i] = \text{hash}[i-1]$ for non-genesis blocks),
- Prefix compliance (each stored hash starts with “00000”), and
- Timestamp monotonicity within each mini-chain.

This means the ledger offers a tamper-evident structure but cannot automatically recompute proof-of-work for past blocks. A stronger guarantee can be achieved in future work by recording nonces and implementing a full-chain automated verifier.

Additionally, OTP delivery relies on external email latency, which sometimes delays lab access. Mining, while responsive at $d=5$, still adds a slight delay to clinical writes.

8.10 Lessons learnt

The role-based access model effectively enforced clinical workflows. OTP challenges assured unauthorised lab writes. Manual ledger verification showed the trade-off between simplicity and automation. This confirmed blockchain's value for tamper resistance.

9 Summary and Conclusion

This project created a tamper-evident EMR prototype that includes a lightweight blockchain at the application layer. Clinical entries are stored as SHA-256 hash-linked blocks with Proof-of-Work at difficulty 5. This setup generates a verifiable timeline across different roles.

Unlike earlier hybrid methods that only store hashes on-chain, this prototype saves the full EMR content, while user and appointment data are kept in a relational database. Role-specific workflows include doctor updates, patient access, OTP-gated lab uploads, and limited third-party reads. Together, these features demonstrate that cryptographic verification can be integrated into clinical workflows without requiring complex multi-node setups.

The importance of the prototype is in demonstrating how integrity-focused EMRs can prevent insider tampering and support the tracking of data origins with few dependencies. However, there are several limitations: mined nonces are not saved, restarts lead to separate mini-chains, and the system operates on a single node without encryption or access logging. Storing complete EMRs on-chain also raises ethical issues, such as GDPR's "right to erasure."

This prototype demonstrates the viability and resilience to insider tampering of a lightweight, integrity-first EMR. Nonce storage, restart handling, and single-node scope without encryption are among the remaining restrictions. Future enhancements that address scalability and GDPR compliance, like TLS, replication, access logs, and privacy-preserving extensions, could bring the system closer to a real-world healthcare deployment.

References

- [1] E. Westphal and H. Seitz, 'Digital and Decentralized Management of Patient Data in Healthcare Using Blockchain Implementations', 2021, *Frontiers Media SA*. doi: 10.3389/fbloc.2021.732112.
- [2] H. J. De Moura Costa, C. A. Da Costa, R. S. Antunes, R. Da Rosa Righi, P. A. Crocker, and V. R. Q. Leithardt, 'ID-Care: A Model for Sharing Wide Healthcare Data', *IEEE Access*, vol. 11, pp. 33455–33469, 2023, doi: 10.1109/ACCESS.2023.3249109.
- [3] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, 'MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain', *J Med Syst*, vol. 42, no. 8, Aug. 2018, doi: 10.1007/s10916-018-0993-7.
- [4] M. A. Al-Khasawneh, M. Faheem, A. A. Alarood, S. Habibullah, and A. Alzahrani, 'A secure blockchain framework for healthcare records management systems', *Healthc Technol Lett*, vol. 11, no. 6, pp. 461–470, Dec. 2024, doi: 10.1049/htl2.12092.
- [5] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, 'MeDShare: Trust-Less Medical Data Sharing among Cloud Service Providers via Blockchain', *IEEE Access*, vol. 5, pp. 14757–14767, Jul. 2017, doi: 10.1109/ACCESS.2017.2730843.
- [6] G. Sabarmathi and R. Chinnaiyan, 'Big Data Analytics Framework for Opinion Mining of Patient Health Care Experience', in *Proceedings of the 4th International Conference on Computing Methodologies and Communication, ICCMC 2020*, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 352–357. doi: 10.1109/ICCMC48092.2020.ICCMC-00066.
- [7] Mettler Matthias, *Blockchain Technology in Healthcare: The Revolution Starts Here*. 2016 IEEE 18th International Conference on e Health Networking, Applications and Services (Healthcom), 2016.
- [8] A. Roehrs, C. A. da Costa, and R. da Rosa Righi, 'OmniPHR: A distributed architecture model to integrate personal health records', *J Biomed Inform*, vol. 71, pp. 70–81, Jul. 2017, doi: 10.1016/j.jbi.2017.05.012.
- [9] A. Dubovitskaya, Z. Xu, S. Ryu, M. Schumacher, and F. Wang, 'Secure and Trustable Electronic Medical Records Sharing using Blockchain', 2017.
- [10] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, 'Healthcare Data Gateways: Found Healthcare Intelligence on Blockchain with Novel Privacy Risk Control', *J Med Syst*, vol. 40, no. 10, Oct. 2016, doi: 10.1007/s10916-016-0574-6.
- [11] T. T. Kuo, H. E. Kim, and L. Ohno-Machado, 'Blockchain distributed ledger technologies for biomedical and health care applications', Nov. 01, 2017, *Oxford University Press*. doi: 10.1093/jamia/ocx068.
- [12] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, 'MedRec: Using blockchain for medical data access and permission management', in *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, Institute of Electrical and Electronics Engineers Inc., Sep. 2016, pp. 25–30. doi: 10.1109/OBD.2016.11.
- [13] T. Nugent, D. Upton, and M. Cimpoesu, 'Improving data transparency in clinical trials using blockchain smart contracts', *F1000Res*, vol. 5, 2016, doi: 10.12688/f1000research.9756.1.

- [14] A. Al Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, and M. S. Rahman, 'Privacy-friendly platform for healthcare data in cloud based on blockchain environment', *Future Generation Computer Systems*, vol. 95, pp. 511–521, Jun. 2019, doi: 10.1016/j.future.2018.12.044.
- [15] W. J. Gordon and C. Catalini, 'Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Driven Interoperability', Jan. 01, 2018, *Elsevier B.V.* doi: 10.1016/j.csbj.2018.06.003.
- [16] Z. Sun, D. Han, D. Li, X. Wang, C. C. Chang, and Z. Wu, 'A blockchain-based secure storage scheme for medical information', *EURASIP J Wirel Commun Netw*, vol. 2022, no. 1, Dec. 2022, doi: 10.1186/s13638-022-02122-6.
- [17] G Sabarmathi and R Chinnaiyan, *Reliable Machine Learning Approach to Predict Patient Satisfaction for Optimal Decision Making and Quality Health Care*. 2019 International Conference on Communication and Electronics Systems (ICCES), 2019.

Appendix

Appendix A

The complete project source code, including server and client implementations, is hosted at:
GitLab Repository:

<https://git.cs.bham.ac.uk/projects-2024-25/dxs477>

The repository contains version history, commits, and issue tracking. This allows for traceability.

Repository contents

- Key blockchain files:
Block.java, readJSON.java, readblockreq.java, blockcserver.java, and StringUtil.java
- Integration of databases:
dbconnect.java and userlogs.json
- Communication and authentication:
ips.java (runtime endpoints) and SendMailExample.java (OTP email)
- Java Swing client applications:
Doctor: doctormenu.java, doctorregister.java, and doctorlogin.java
Patient: patientmenu.java, patientlogin.java
Lab Technician: labmenu.java, lablogin.java
Third-party/guest: guestlogin.java, guestreport.java
- Designs and documentation:
Implementation_.docx, Design.odt, Literature_review.odt, and
BLOCKCHAIN_project_report.docx (draft)
- Diagrams: Screenshots 2025-07-19_132249.png (system design), 2025-07-19_132249.png (use case), and 2025-08-10_134327.png (implementation)

Instructions to run the system

1. Clone the repository:

```
git clone https://git.cs.bham.ac.uk/projects-2024-25/dxs477.git
```
2. Start MySQL and import the schema from /database/schema.sql.
3. Compile and run the blockchain server:

`javac blockserver.java` and `java blockserver`

4. Launch client applications (Doctor, Patient, Lab, Guest) from their respective .java files.
5. For lab uploads, OTPs are sent via the configured SMTP in `SendMailExample.java`.
6. View tamper-evident ledger updates through `readJSON.java` and `userlogs.json`.

Appendix B

Declaration

I certify that this project is my own original work. Code debugging support was assisted by ChatGPT (version 4.0), and report editing for clarity and consistency was supported by ChatGPT and Grammarly. All outputs were reviewed and edited by me to accurately reflect my own contributions.

Name: Deepthi Sarvamangala Mouli

Student ID: 2801976

Date: 01/09/2025

Appendix C

This table supports Section 2.6 and compares hybrid blockchain EMR designs with the implemented prototype.

Aspect	Prior “hybrid” designs	This prototype
Storage placement	EMR off-chain; hashes/metadata on-chain	EMR on-chain; DB for users & appointments
Consensus	Permissioned consensus (RAFT/PBFT), multi-node ops	No cluster; PoW (difficulty = 5) as local cost, single node
Integrity check	Fetch content and compare to the on-chain digest	Self-contained: rehash fields, verify previous hash + PoW prefix
Rewrite cost	Low if off-chain store compromised	High: must re-mine the modified block and all successors
Ordering	Often DB/contract timestamps	Append order on chain; timestamps for sanity
Availability goal	High (multi-node)	Integrity-first; availability out of scope

Table 5: Prior hybrids vs this prototype

Appendix D

Requirement Gathering (Interviews & Personas)

Interviews

Formal interviews were conducted with a doctor and a patient to enhance the requirements gathering process. These interviews helped in getting real-world pain points and understanding the expectations of the users, and then shaped the functional and non-functional requirements of the EMR sharing system.

Interview 1: Dr. Keerthi - General Physician

Dr. Keerthi expressed the difficulties she faced while accessing the patient's historical reports during clinical consultations. She often refers to the physical documents or insecurely stored reports on email or WhatsApp, which are inefficient and prone to data loss or privacy risks. She emphasised the importance of on-demand access to patient records during times of emergencies, but the reports are being lost or are unable to access them immediately.

Dr. Keerthi welcomed the idea of logging into the patient report using the doctor's login ID and password, which forms a balance between security and usability. She even expressed a strong interest in having the audit to check if the records were accessed and by whom, which maintains accountability.

Interview 2: Ms. Adi – Patient with chronic condition

Ms. Adi maintained a physical folder to manage the medical records and some on the smartphone gallery or downloads. She explained it as a miserable task to keep those records safe without losing them or storing them securely. She appreciated the idea of a digital platform where all the medical records are updated by the doctor and all the medical history is stored securely in a system, and accessing them by a third party would require the patient's approval.

She expressed concern that sharing the medical data without her consent would lead to the misuse of data. She was happy to know that my proposed system would be storing the access logs and was in favour of immutable records that cannot be changed by third parties, but could only view them upon the patient's approval. She mentioned the clean and well-structured interface so that even non-technical users would be able to use the application easily.

Therefore, the interviews provided a clear insight into the requirements of the project developed, like:

- OTP-based access control to ensure explicit patient consent for every record access.
- Blockchain transaction logging to provide transparency and traceability of data access events.
- Role-based permissions to prevent unauthorised viewing or editing of records.
- Patient-centric control model, where patients initiate and approve all access requests.

Personas

Before starting to build the application, it is required to understand the perspective of different types of people using it. Each person plays different roles, has technical familiarity, and expectations. The personas below demonstrate the point of view of various users, like the patient, doctor, lab technician and the third-party who engage with the system for different purposes. Through each persona, the background, goals, challenges, and how the proposed system is used help in the decision-making of usability, access control, and feature prioritisation.

The following personas help in understanding the requirements of the system in the real world and ensure that the system design supports diverse workflows and access patterns.

Table 6: Persona 1- Patient

Name	Adi
Role	Record owner
Age	55
Background	Working professional with chronic illness
Technical familiarity	Moderate
Goals	<ol style="list-style-type: none"> 1. Log in and view her medical records 2. Control who can access her data 3. Receive access requests and approve/reject them easily (e.g., via OTP)
Challenges	<ol style="list-style-type: none"> 1. Concerned about privacy and unauthorised access

	<ol style="list-style-type: none"> 2. Wants an easy-to-use interface 3. May not understand technical terms
How the proposed system works	<ol style="list-style-type: none"> 1. Logs in to go through the prescriptions or lab results 2. Approves record access for the third-party (other than the doctor) and lab technicians with the OTP 3. Reviews access history via blockchain audit log

Table 7: Persona 2- Doctor

Name	Dr. Keerthi
Age	30
Background	Tech-comfortable, uses digital systems in the clinic
Tech familiarity	High
Goals	<ol style="list-style-type: none"> 1. Quickly view patient EMRs whenever required 2. Make informed medical decisions based on historical records stored 3. Make sure the prescriptions are accessed only by him with his login details
Challenges	<ol style="list-style-type: none"> 1. Limited time during consultations 2. Needs immediate access in emergencies 3. Wants seamless integration without login delays

How the proposed system works	<ol style="list-style-type: none"> 1. Logging with her ID and password 2. Reads the approved records and uploads a consultation summary
-------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 8: Persona 3- Friend/Caretaker (Third-party access)

Name	Niha
Age	28
Background	Not in the healthcare profession
Tech Familiarity	Moderate
Goals	<ol style="list-style-type: none"> 1. Access EMR when the patient gives explicit consent
Challenges	<ol style="list-style-type: none"> 1. Doesn't understand medical terminology 2. Needs temporary, limited access 3. The third party may change data that is stored, leading to security risks 4. Requires a clear interface with minimal confusion
How the proposed system works	<ol style="list-style-type: none"> 1. Gets temporary OTP-based access granted by the patient 2. View lab results or prescriptions 3. Cannot upload or edit data

Table 9: Persona 4- Lab Technician

Name	Anu
Age	32
Background	Works at a hospital
Tech familiarity	High for lab systems but low for blockchain concepts

Goals	<ol style="list-style-type: none"> 1. Upload lab reports to a patient's EMR when permitted 2. Ensure reports are securely sent without tampering
Challenges	<ol style="list-style-type: none"> 1. Needs clear prompts and role-based permissions
How the proposed system works	<ol style="list-style-type: none"> 1. Gets access to upload the reports only when the patient gives access 2. Uploads lab test results for a patient 3. Cannot view unrelated records or patient history

Appendix E

Accessibility Requirements

This appendix includes detailed reference tables and proof of how the system meets accessibility requirements (WCAG 2.1 AA), along with age-specific accessibility for children and seniors. It also provides testing notes and compliance references.

1. Perceivable

No	Accessibility Requirement
1.1	Ensure content is readable without loss of information, even when zoomed
1.2	Information must be structured with proper use of headings and form labels (login fields, registration fields have labels like Name, Password)
1.3	Buttons and text follow high-contrast schemes so that even users with low vision can distinguish

Table 10: WCAG 2.1 (AA) - Perceivable

2. Operable

No	Accessibility Requirement
2.1	All core functionality, like the form submission and access toggling, must be operable using only a keyboard (All menus, login forms, and blockchain actions can be done with Tab/Enter, no mouse required)
2.2	Provide clear navigation indicators and allow users to locate their current position in the app
2.3	Use clear and descriptive headings and labels throughout the interface

Table 11: WCAG 2.1 (AA) - Operable

Table 12: WCAG 2.1 (AA) - Understandable

3. Understandable

No	Accessibility Requirement
3.1	Content should use simple, familiar language (e.g., “Append to Ledger” should be mentioned as “Save to chain”)
3.2	App behaviour should be predictable (clicking a button always opens the intended form, no hidden actions).

3.3	Provide helpful error messages and correction suggestions when incorrect inputs are given (e.g., “Please enter patient ID” instead of generic “Error”).
-----	---------------------------------------------------------------------------------------------------------------------------------------------------------

Table 13: WCAG 2.1 (AA) - Robust

4. Robust

No	Accessibility Requirement
4.1	Work on different screen sizes and Java-supported systems
4.2	All UI components (forms, toggles, links) should have identifiable and programmatically defined roles in Java Swing
4.3	Double confirmation for deleting or sending records

Table 14: Requirements for Children

No	Accessibility Requirement
2.1	Buttons and form fields should be big enough for children to click/tap easily
2.2	The UI flow can match a simple story: Book Doctor → Visit Lab → See Report
2.3	Works the same whether on a school computer, tablet, or parent’s laptop
2.4	Children may struggle with typing so use drop-down or checkbox where possible
2.6	Use of simple, familiar terms (e.g., Doctor Login, Patient Register, Lab Menu)

Table 15: Requirements for Seniors

No	Accessibility Requirement
2.1	Text and buttons must support a large-screen display with scalable font sizes.
2.2	All interactive elements, like buttons, form fields, should be large enough for easy clicking
2.3	The app should follow a linear and logical menu structure, avoiding hidden options or complex gesture navigation.
2.4	For critical actions, confirmation is required (e.g., patient registration, deleting a record)
2.5	Consistent colour contrast (NetBeans default buttons with text have high contrast)
2.6	Use of simple, familiar terms (e.g., Doctor Login, Patient Register, Lab Menu)
2.7	No strict timeouts in forms, so elderly users won’t get logged out too quickly

Accessibility requirements were tested using Java Swing UI components. This was done at various zoom levels and with different input methods.

Appendix F

Project Management Artefacts

Figure 10 shows the planned timeline for research, design, development, and evaluation over 12 weeks. The tasks were arranged to respect dependencies, such as gathering requirements before coding.

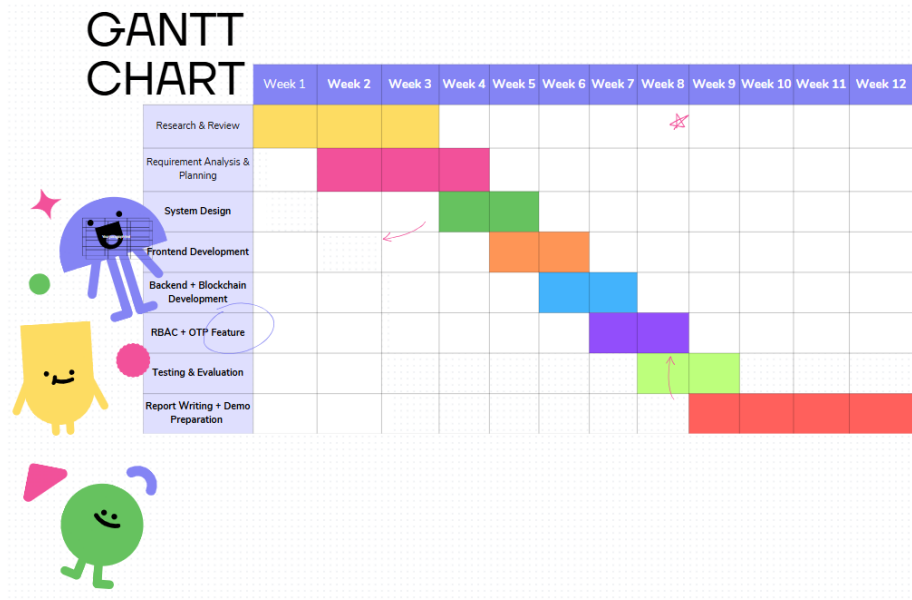


Figure 10: Project Gantt Chart

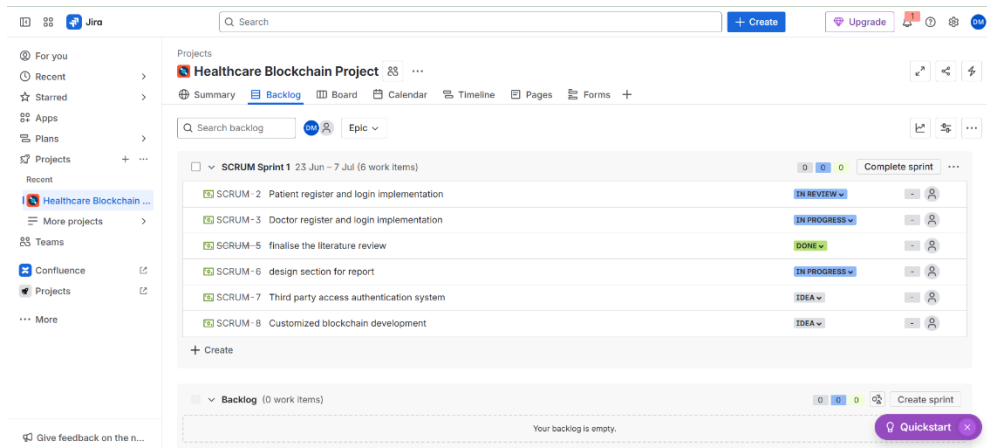


Figure 11: Jira Scrum Board (Sprint Management – Work-in-Progress)

Figure 11 shows the Jira Scrum board used to track sprint items like login implementation, RBAC feature integration, and blockchain consensus design. Tasks are assigned and tracked until they are completed. The screenshot was taken during the development phase.

Appendix G

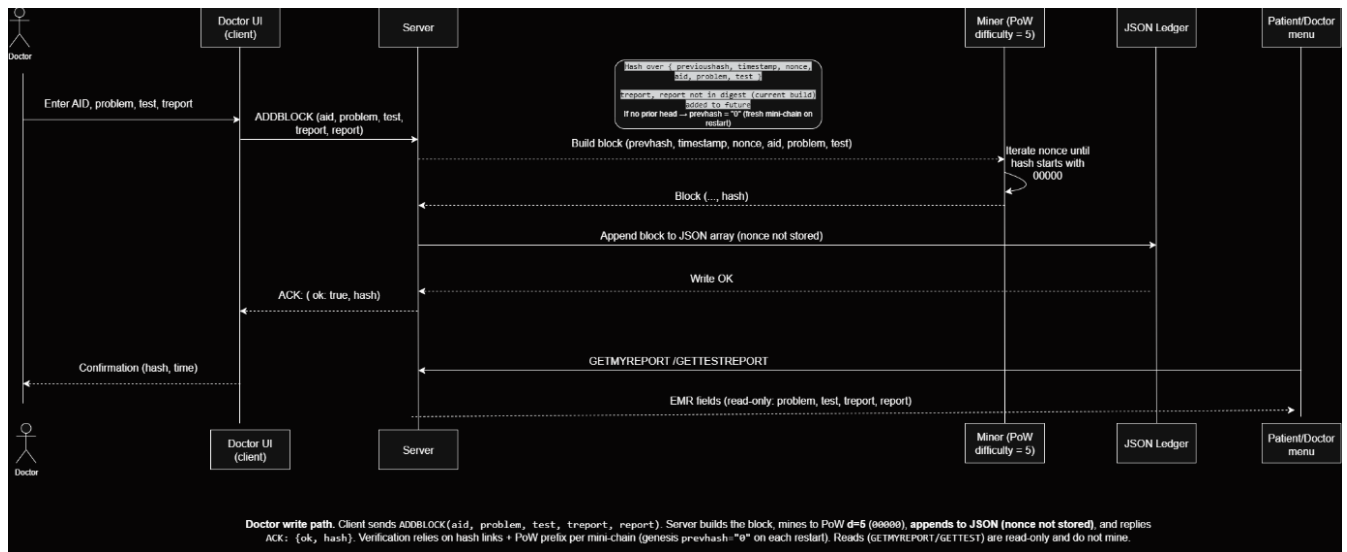


Figure 12: Sequence Diagram of Doctor Write Path

String	Typical caller	Request payload	Server reply	Purpose
ADDBLOCK	Doctor (via sendblock.addblock)	int aid, String problem, String test, String treport, String report	"SUCCESS" or error	Appends a new mined block (PoW difficulty=5) with previoushash = current head
GETMYREPORT	Patient (patientmenu), Guest/Third-party (guestreport)	int aid	"FOUND", then problem, test, treport, report (Strings); or "NOTFOUND"	Read a complete EMR entry for an appointment and fetch it for guest viewing
GETTESTREPORT	Doctor (doctormenu)	int aid	"FOUND", then test, treport, report (Strings); or "NOTFOUND"	Doctor views lab test + doctor report.

GETTEST	Lab tech (labmenu)	int aid	"FOUND", then test, treport (Strings); or "NOTFOUN D"	Lab fetches test name + current short test report.
UPDATETEST REPORT	Lab tech (labmenu)	int aid, String treport	"SUCCESS" or error	Updates treport for that aid Fetch EMR for Appointment ID
UPDATEDOCT ORREPORT	Doctor (doctormenu)	int aid, String report	"SUCCESS" or error	Updates report (doctor prescription/diagnosi s).

Table 16: Socket Protocol Message Contract

Appendix H

Consensus and Storage Design Decisions

This appendix explains the reasons for selecting a custom Proof-of-Work chain with a difficulty of 5.

Options	Pros	Cons	Suitability here
Database only	Simple ops, great queries	Policy-based; an insider can rewrite	Not suitable
Hash-chain only	Linkage, no extra infrastructure	Cheap to recompute the forged tail	Not suitable
Signatures-only	Authorship proof	No chain-ordering integrity	Not suitable
RAFT/PBFT (permissioned)	Availability, federation	Heavy operational overhead	Not suitable for my single-node prototype
JSON append-only ledger	Self-verifiable with just the file. It has a simple backup. There is no dependency on a database or cluster. It works with a single-writer server.	Manual tooling where care is needed for atomic writing	Chosen for integrity
Custom chain + PoW(d=5)	Tampering is costly, while manual verification remains trivial. Works fine in a node, dependency-light setup	Ledger grows larger, and you must manage privacy carefully since all clinical writes are on-chain	Chosen

Table 17: Comparison of Storage and Consensus Options

Appendix I

Build and Deployment

I.1 Prerequisites

- JDK: Java SE 8 or 11LTS
- My SQL server: reachable at the host configured in ips.java (default 127.0.0.1), DB name emrsharing
- JDBC Driver: MySQL connector/J on the classpath (used the legacy driver class com.mysql.jdbc.Driver)
- ips.java sets blockchainip="127.0.0.1", port=3000, mysqlip="127.0.0.1"
- SMTP: Gmail account with App Password (for OTP emails) and uses smtp.gmail.com:465 (SSL) for sending the OTP mails

Components	Functions
JLabel	Used for text like “Patient ID”, ” Name”, titles, and section headers
JTextFields	Used for entering the text inputs, like entering the name, ID
JPasswordField	Used for entering the password in the password and confirmation password fields
JComboBox	Drop-down selection for choosing doctors or appointment types
JButton	For the actions like Register, Login, View Appointments, Book Appointment, and Log out, where an ActionListener is linked for event handling
JTabbedPane	Provides tab-based navigation between features like viewing appointments, booking appointments, and medical records
JTable	Displays the appointment and record data retrieved from the database, which is linked with DefaultTableModel for increasing the table rows
JOptionPane	Display the pop-up messages like errors, success, and warnings
JDateChooser	Selecting a date from the calendar widget for booking an appointment
JFrame	Main screen insertion (like register, patientlogin, patientmenu)
JScrollPane	Allows scrolling when there are large datasets

Table 18: UI Swing components used

I.2 Additional JAR dependencies

- jcalendar-1.4.jar – calendar/date-picker UI components (used in booking appointment)
- json-simple-1.1.jar – JSON serialisation/deserialization for blockchain ledger.
- mail.jar – JavaMail library for sending OTP emails.
- mysql.jar – MySQL JDBC driver

I.3 Build and start

- Open the project and add the libraries: MySQL Connector/J, json-simple, javax.mail.
- Start the server by running blockchainServer.blockcserver. This opens a Swing server window and binds the TCP ServerSocket to port 3000.
- Start clients by running the following UIs from NetBeans:
 - doctorlogin leads to doctormenu
 - patientlogin leads to patientmenu
 - lablogin leads to labmenu (uses SendMailExample for OTP)
 - guestlogin/guestreport allows for third-party read-only access
- Ensure the working directory is writable. The ledger file userlogs.json is created and read here by readblockreq/readJSON.

I.4 Quick demo flow

- DB ready: Create the schema emrsharing and tables as per users (patient, doctor, lab technician, third-party) and appointments. Add one user for each role and one appointment for the patient.
- Start the server (blockcserver). It listens on port 3000 (from ips.java).
- The doctor creates an EMR block: Log in as the Doctor after registration is done. Then enter aid, problem, test, treport/report, then submit.
 - The server mines the block at difficulty 5 (prefix via StringUtil.applySha256) and appends it to userlogs.json.
- The patient views: Once registration is done, log in as patient, then view EMR by aid.
- Lab upload with OTP: Login as Lab, request an OTP (an email is sent via SendMailExample to the lab's registered Gmail), enter the OTP, then upload treport/report. The mined block is appended on-chain. (The OTP is temporary and not stored in the database.)

- Third-party read-only: Open guestlogin/guestreport, enter Appointment ID and DOB, then read the EMR in read-only mode. Any write attempts are rejected.
- Manual integrity check (manual verification): Open userlogs.json. For consecutive blocks, ensure previoushash[i] equals hash[i-1] and that hash starts with the specified prefix (PoW=5).

Note: timestamps should not decrease during a session. A new mini-chain starts with each server restart (genesis uses "0" in previoushash by design).

I.5 Troubleshoot the errors

- Port conflict: If port 3000 is busy, change ips.port on the server and clients.
- MySQL errors: Check Connector/J on the classpath, the DB name emrsharing, and the credentials in dbconnect.java.
- SMTP/OTP: Gmail often requires an app password for SMTP over SSL on port 465. You must allow network egress to smtp.gmail.com.
- Ledger location: userlogs.json is in the server's working directory. Keep it with your build artefacts when you demo or zip the project.
- Restart behaviour: Each restart starts a new mini-chain (fresh genesis) by design. Ordering and verification happen per run.

Appendix J

Sl.no	Scenario	Pre-conditions	Steps	Expected
1	Doctor updates report	Doctor logged in; existing aid	Edit report → submit UPDATEDOCTORREPORT	Update reflected in view (note: if field not hashed, no re-mine)
2	Patient registers, books, and views	—	Register → book appointment → open EMR view	Row created in appointments (DB); EMR view by aid works
3	Invalid login (any role)	User exists	Enter the wrong password	Login denied
4	Doctor adds with missing fields	Doctor logged in	Submit ADDBLOCK with empty problem/test	Rejected with validation error
5	Doctor updates non-existent aid	Doctor logged in	UPDATEDOCTORREPORT with unknown aid	NOTFOUND; no changes
6	Lab OTP with wrong code	Lab logged in	Request OTP → enter incorrect code	Denied; no write; OTP remains transient
7	Third-party wrong DOB	Appointment exists	Correct Appointment-ID + wrong DOB	Denied; no data returned
8	Third-party wrong Appointment-ID	—	Invalid Appointment-ID + valid DOB	NOTFOUND/Denied; no data
9	Third-party attempts write	Third-party authenticated	Try ADDBLOCK/update op	Rejected (read-only role)

10	Ledger appends I/O failure	File locked or disk full	Doctor submits ADDBLOCK	No partial write; error to UI; server continues
11	SMTP failure during OTP	Mail server unavailable	Lab requests OTP	User sees a failure message, and no OTP is issued
12	Back-date/replay attempt	JSON file accessible	Manually insert the duplicate old block later in the file	Manual check flags ordering/timestamp sanity
13	Doctor creates EMR block	Doctor logged in; valid aid	Login → enter aid, problem, test, treport/report → submit	Server mines (d=5); new block visible in recent for aid
14	Lab upload with OTP	Lab logged in; registered email	Request OTP → receive email → enter OTP → upload treport	On success, write stored on-chain; OTP not persisted
15	Third-party read-only access	Appointment exists	Enter Appointment-ID + DOB → view	Read-only EMR shown; any write op rejected
16	Concurrent actions contention	Two clients active	Doctor ADDBLOCK while Patient views	Both complete; no deadlock; consistent view

Table 19: Extended Functional Correctness Scenarios

Appendix K

Test	Manipulation	What is manually verifiable and is observed
Break link	Edit previoushash of block k+1 so it \neq hash[k]	Link check fails at k+1
Break PoW prefix	Edit a block's hash so it no longer starts with 00000	Prefix check fails
Replay/back-date (order sanity)	Insert/duplicate an old block later, or back-date the timestamp	Timestamp sanity and per-run ordering flag inconsistencies
Known gap	Change only treport/report (not in digest)	Links & prefix still pass (documented limitation)

Table 20: Tampering Experiments on Ledger Integrity

```

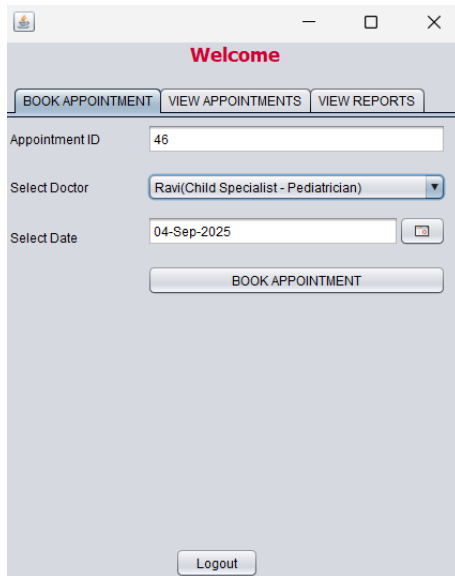
"block": {
  "problem": "ear pain, throat pain",
  "test": "ear scan",
  "previoushash": "0",
  "treport": "an obstacle is found",
  "report": "Dont rub your ears, consult tomorrow morning",
  "aid": 34,
  "hash": "00000f2a9b045598655d8eb5b95de3402299420b575943f799614e1af06a4784",
  "timestamp": 1755421583719
},
{
  "block": {
    "problem": "red spots on hands and feet",
    "test": "blood test",
    "previoushash": "00000f2a9b045598655d8eb5b95de3402299420b575943f799614e1af06a4784",
    "treport": "infection found",
    "report": "foot and mouth disease detected",
    "aid": 35,
    "hash": "00000b36c51f155919502dd1368d5a811ac252112a3d956f31bf185a52f2bfcf",
    "timestamp": 1755422569788
  },

```

Figure 13: Evidence of Blockchain Integrity

Appendix L

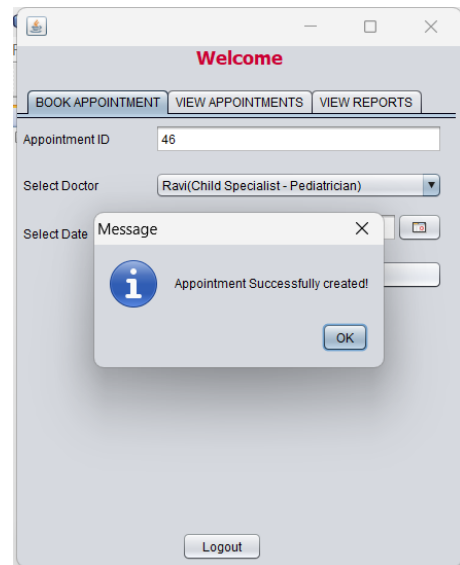
System Demonstration Screenshots



The screenshot shows a web application window titled "Welcome". It has three tabs: "BOOK APPOINTMENT", "VIEW APPOINTMENTS", and "VIEW REPORTS". The "BOOK APPOINTMENT" tab is active. The form contains the following fields and controls:

- Appointment ID:** A text input field with the value "46".
- Select Doctor:** A dropdown menu with "Ravi(Child Specialist - Pediatrician)" selected.
- Select Date:** A date input field with "04-Sep-2025" and a calendar icon.
- BOOK APPOINTMENT:** A button.
- Logout:** A button at the bottom.


Figure 14: Appointment booking screen



The screenshot shows the same "Appointment booking screen" as Figure 14, but with a modal message box displayed. The message box has a title bar "Message" and contains the following:

- Message:** "Appointment Successfully created!"
- OK:** A button to close the message box.

Figure 15: Appointment successfully created



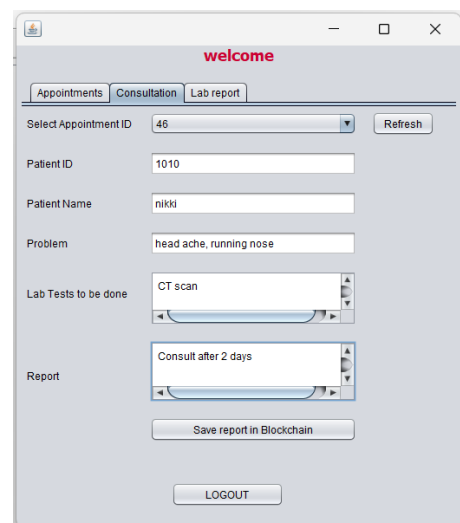
The screenshot shows a web application window titled "welcome". It has three tabs: "Appointments", "Consultation", and "Lab report". The "Appointments" tab is active. The form contains the following fields and controls:

- Refresh:** A button.
- Table:** A table with the following data:

Appointment ID	Patient ID	Date	Status
46	1010	Thu Sep 04 11:49:...	PENDING

- Accept:** A button.
- Cancel:** A button.
- LOGOUT:** A button at the bottom.

Figure 16: Doctor dashboard-pending appointment



The screenshot shows a web application window titled "welcome". It has three tabs: "Appointments", "Consultation", and "Lab report". The "Consultation" tab is active. The form contains the following fields and controls:

- Select Appointment ID:** A dropdown menu with "46" and a "Refresh" button.
- Patient ID:** A text input field with "1010".
- Patient Name:** A text input field with "nikki".
- Problem:** A text input field with "head ache, running nose".
- Lab Tests to be done:** A text input field with "CT scan".
- Report:** A text input field with "Consult after 2 days".
- Save report in Blockchain:** A button.
- LOGOUT:** A button at the bottom.

Figure 17: Doctor consultation screen

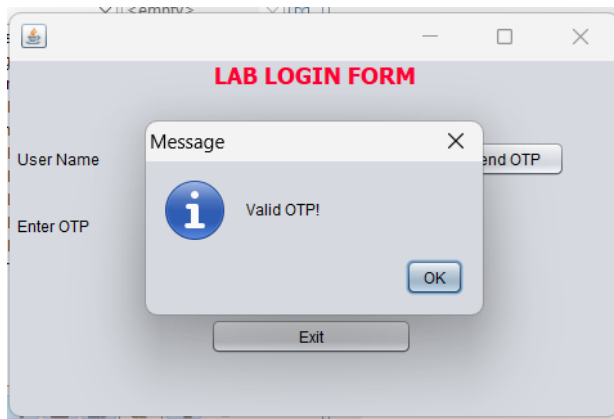


Figure 18: Lab technician login with OTP validation

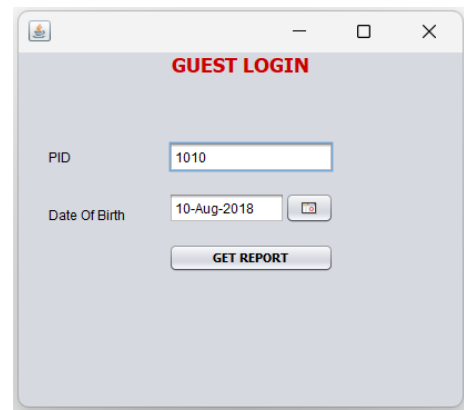


Figure 19: Guest login screen

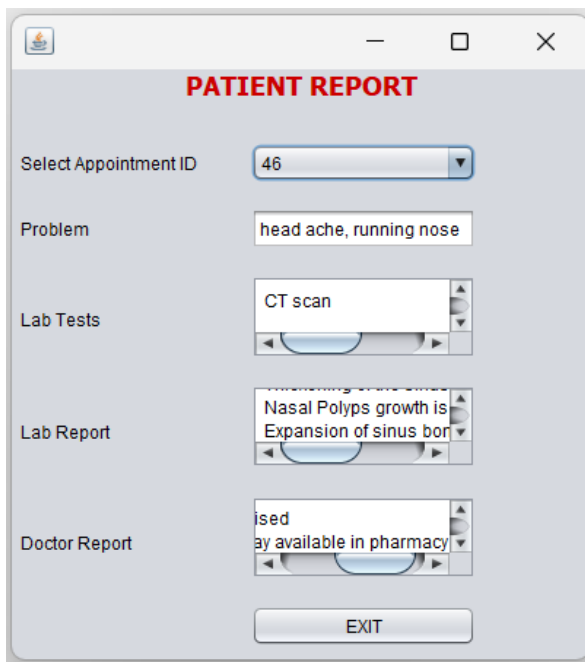


Figure 20: Patient report view

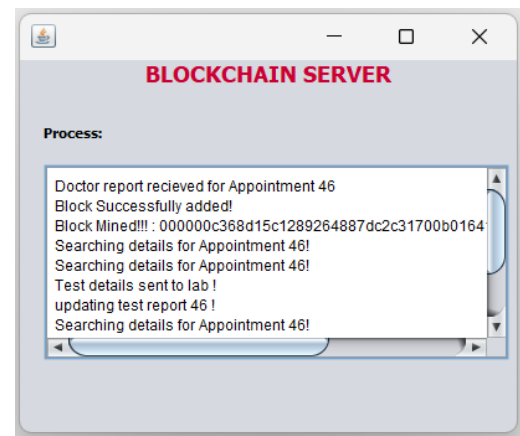


Figure 21: Blockchain server console



Figure 22: JSON ledger showing hash connection and new chain formation

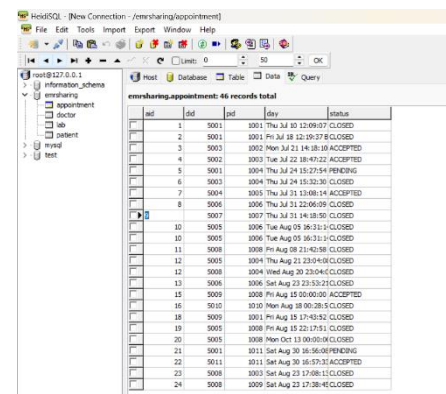


Figure 23: Appointment metadata stored in HeidiSQL

Appendix M

Project resources and diagram links

This appendix provides access to external resources for diagrams that were created during the project.

- Sequence diagram (Doctor Write Path): https://drive.google.com/file/d/1E9KePlyYfS-fyci4qPqGc_bd3t79utb0/view?usp=sharing .Created using diagrams.net (draw.io).

- Class diagram (Blockchain-based Medical Data Management System):
<https://drive.google.com/file/d/1nLjqGQgRRkZ1qou4iMsa4uxuDSYc2Zkq/view?usp=sharing> .
Created using diagrams.net (draw.io).

- System architecture, UML Diagram, Blockchain Structure:
https://www.canva.com/design/DAGtPW2aVsc/r0LykYm0GvButaYAbhOCeQ/edit?utm_content=DAGtPW2aVsc&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton