DECENTRALIZED MEDICAL RECORD MANAGEMENT USING BLOCKCHAIN TECHNOLOGY

Hanane ZAOUI, Hamza HAFDAOUI , Salma AMGAROU

Master's of Artificial Intelligence & Data Science, Computer Science Department
Faculty of Science And Technology, Tangier, Morocco
University of ABDELMALEK ESSAADI, Tétouan, Morocco
Hamza.hafdaoui@etu.uae.ac.ma
Salma.amgarou@etu.uae.ac.ma
Hanane.zaoui@etu.uae.ac.ma

Keywords: *Blockchain Technology, Decentralized Applications (DApps), Decentralized Medical Records, Smart Contracts, Ethereum, InterPlanetary File System (IPFS), Web3.py, Role-Based Access Control (RBAC), Medical Record Security*

Abstract

This paper presents a decentralized application (DApp) for secure and efficient medical record management utilizing Ethereum blockchain and InterPlanetary File System (IPFS). By leveraging smart contracts developed in Solidity, this project ensures confidentiality, integrity, and traceability of medical data while promoting transparency and decentralization. The system employs a Python-based graphical interface to facilitate interaction with the blockchain, enabling functionalities such as patient registration, medical record updates, and access control. Performance evaluations demonstrate the efficacy of this approach in addressing the challenges of traditional centralized systems.

1. INTRODUCTION

1.1 Problem Statement:

The management of medical records is critical in modern healthcare but is often hindered by the vulnerabilities of traditional centralized systems. These include susceptibility to cyberattacks, lack of transparency, and inefficiencies in access control. As medical data breaches grow increasingly common, there is an urgent need for secure, decentralized solutions that address these issues effectively.

1.2 Proposed Solution

This project introduces a blockchain-based DApp that uses Ethereum smart contracts to securely manage medical records. The proposed system ensures:

- *Confidentiality*: Medical records are encrypted and securely stored in IPFS, with access restricted through cryptographic keys.
- *Integrity:* Blockchain immutability prevents unauthorized alterations to data.
- *Decentralization:* Eliminating reliance on a central authority enhances resilience.
- *Transparency and Traceability:* Detailed audit logs recorded on the blockchain provide a comprehensive history of all transactions.

1.3 Scope

The DApp includes:

- **Smart contracts** for patient and doctor management, medical record handling, and auditing.
- **Front-end interface** built with Python and customtkinter.
- **Backend integration** leveraging Web3.py for blockchain interactions and IPFS for record storage.

2. STATE OF THE ART

2.1 Existing Solutions

Traditional healthcare systems predominantly use centralized databases, which are prone to data breaches and administrative inefficiencies. Blockchain-based alternatives address these issues but often lack the scalability or seamless user experience required for widespread adoption. The usual implementation of blockchain applications, particularly for transaction validation, is through web applications rather than desktop apps. Web apps provide seamless connectivity with wallets like MetaMask, leveraging browser extensions for efficient transaction handling. This approach ensures better feasibility due to standard APIs and broader user adoption of web-based tools.

Examples of existing blockchain-based solutions include:

- **MedRec:** A blockchain system designed to manage medical records while ensuring patient control over their data.
- **Patientory:** A healthcare-focused DApp leveraging blockchain and IPFS for data storage and patient-centric control.
- **HealthChain:** A cross-platform solution integrating blockchain with modular accessibility for healthcare providers and patients.

These solutions highlight the potential of blockchain but also underline the need for user-friendly interfaces and integration mechanisms for practical usage.

2.2 Technology Selection

- *Ethereum Blockchain:* Chosen for its robust ecosystem, smart contract capabilities, and extensive documentation. Ethereum has been widely adopted in various decentralized applications (DApps) due to its support for programmable smart contracts and a vast developer community. Examples include the MedRec system, which uses Ethereum for patient-centric medical record management, enabling patients to control access to their records.

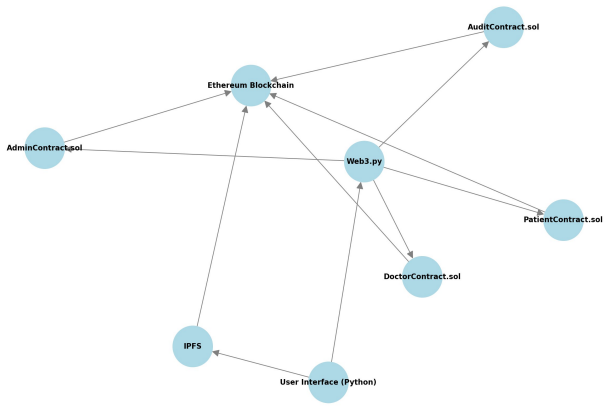
- *IPFS:* Used for storing large medical files, ensuring decentralized and efficient data retrieval. In projects like Patientory, IPFS complements blockchain by enabling the storage of medical data off-chain while maintaining data integrity through cryptographic hashes stored on the blockchain.

- *Python:* Selected for building an intuitive and cross-platform graphical interface. Python's libraries, like Web3.py and frameworks such as customtkinter, provide seamless integration with blockchain networks and a user-friendly development experience, as seen in solutions like **HealthChain**, which prioritizes cross-platform access*ibility and modularity for healthcare applications.*

3. SYSTEM ARCHITECTURE

3.1 Overview

The system integrates blockchain technology, IPFS, and a Python-based user interface to create a decentralized, user-centric medical record management solution. The architecture is designed to ensure seamless interaction between components, as illustrated in the uploaded project structure diagrams.



3.2 Components

3.2.1 Smart Contracts

- *PatientContract:* Manages patient registrations, profile updates, and record permissions. Functions include registerPatient, which securely stores patient information on the blockchain, and **addMedicalRecord**, which links IPFS-stored records with the blockchain.

- *DoctorContract:* Handles doctor registrations and access permissions. Key functions include registerDoctor for securely onboarding healthcare professionals and grantAccess, which authorizes doctors to view specific patient records.

- *AdminContract:* Provides administrative control over the system. Admins can add or revoke roles, monitor system activities, and validate permissions. Key functions include **addAdmin** to register admins and **updateSystemParameters** to manage configurations.

- *AuditContract:* Logs all interactions for traceability and accountability. The **logAction** function ensures a tamper-proof record of all operations, including access and modifications, and **updateSystemParameters** to modify key operational settings. **AdminContract** plays a critical role in maintaining overall governance of the system while integrating seamlessly with other contracts.

3.2.2 Frontend

Developed using Python's customtkinter for a modern UI, supporting multi-role functionalities such as admin, doctor, and patient panels.

3.2.3 Backend

- *Blockchain Interaction:* Enabled via Web3.py for Ethereum smart contracts. Example: the **fetch_logs** function retrieves audit records from the blockchain for display in the frontend.

- *Storage Mechanism:* Medical records are stored in IPFS, with file hashes logged on the blockchain.

3.3 Data Flow

3.3.1 Registration:

- Patients, doctors, and admins register via the Python interface, and details are stored on the blockchain.

3.3.2 Medical Record Management:

- Records are uploaded to IPFS, and the generated hash is stored on the blockchain for verification.

3.3.3 Access Control:

- Patients can grant or revoke access to their medical records by inputting a doctor's wallet address, as illustrated in the screenshots.

3.3.4Audit Trail:

- Every action is logged on the blockchain, ensuring a tamper-proof history.

4. IMPLEMENTATION

4.1 Smart Contract Design

- ***PatientContract:*** Implements functions such as registerPatient and **addMedicalRecord**. The contract enforces strict access controls, allowing only authorized users to interact with sensitive data.

- ***DoctorContract:*** Provides functionality for registering doctors and granting them permissions to access patient data.

- ***AdminContract:*** Enables administrative oversight, including role management and system updates. Example: **addAdmin** to register administrators and validateAccess to verify permissions.

- ***AuditContract:*** Maintains a detailed record of all transactions and interactions for transparency.

Key Example: The authorizeContract function in AuditContract ensures that only verified contracts can log actions, adding an additional layer of security.

4.2 Integration

4.2.1 Contract Deployment:

- Smart contracts are deployed on Ganache, a local Ethereum blockchain environment.

- Deployment utilizes Truffle for migration and management of contracts.

4.2.2 Python-Blockchain Interaction:

- *Web3.py* facilitates communication between the Python frontend and Ethereum.

- *Example:* The **register_patient** function in the frontend directly interacts with the blockchain by calling **registerPatient** in **PatientContract.**

4.2.3 IPFS Integration:

- Medical records are uploaded to a local IPFS node via Python scripts. The resulting hash is stored on the blockchain, ensuring data integrity

- ***Speech Recognition:*** Utilized Google Speech Recognition API to listen to user commands.
- ***Language Model Integratio***n: Incorporated LLaMA 3.1 Versatile Groq, a large language model (LLM), to generate clear responses and assist the user in-store.
- ***Real-Time Object Recognition***: Integrated the optimized YOLOv8 Medium model for real-time product identification.

4.3 Frontend Design

4.3.1 Role-Based Interfaces:

- Separate panels for admin, doctors, and patients, designed using **customtkinter**.

- *Example:* The patient dashboard enables users to upload medical records, grant/revoke access, and view logs, as shown in the screenshots.

4.3.2 Features:

- Patient and doctor registration.

- Medical record upload and retrieval.

- Role-based access control **(grant/revoke permissions).**

- Profile management.

4.3.3 Storage Mechanism

IPFS Integration:

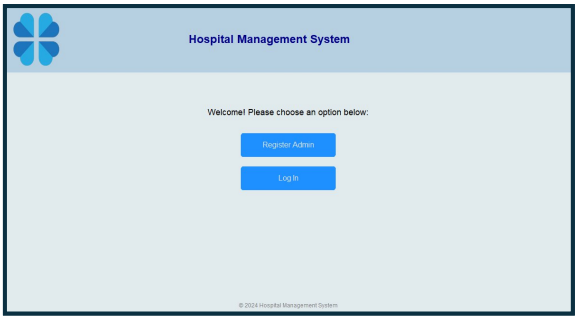
- Files are uploaded to a local IPFS node. The resulting hash is stored on the blockchain, ensuring data integrity.

- Example: The **upload_to_ipf**s function automates the file upload process and returns the unique IPFS hash.

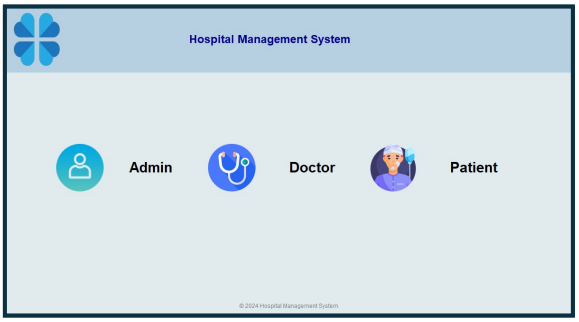
5. RESULTS

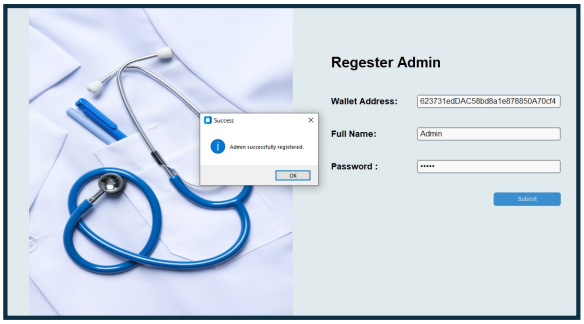
5.1 Demonstration

5.1.1 Role Selection Page

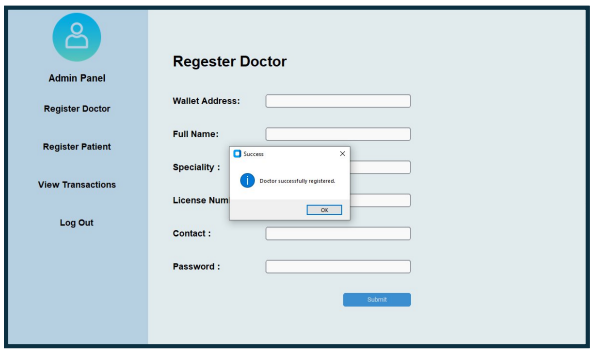


5.1.2 Register Admin

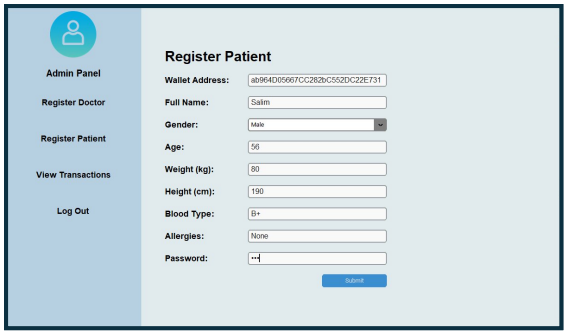




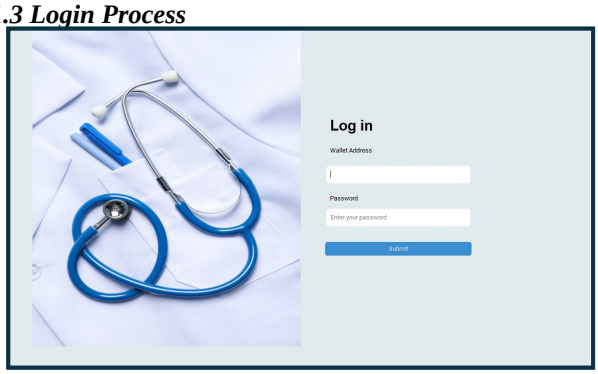
5.1.3 Register Doctor

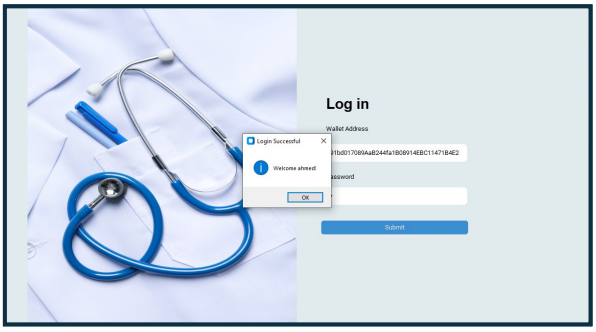


5.1.4 Register Patient

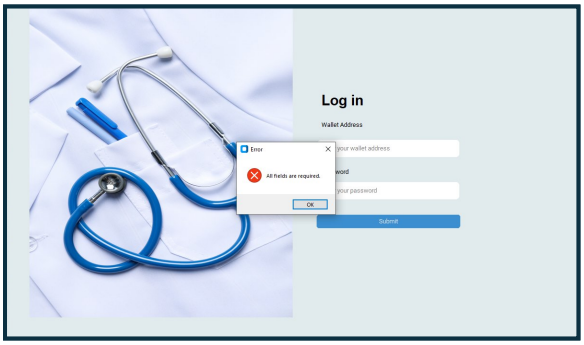


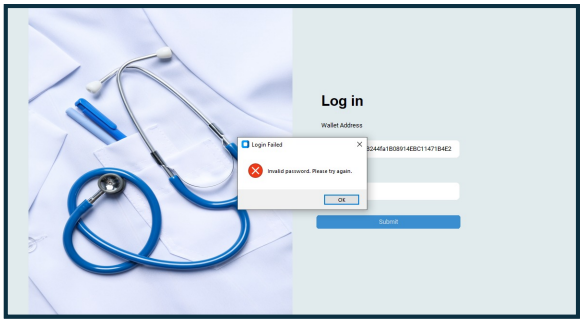
5.1.3 Login Process



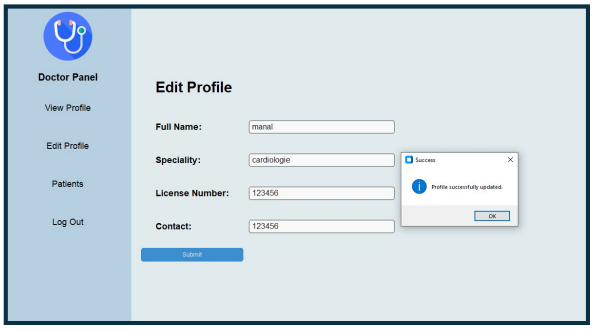


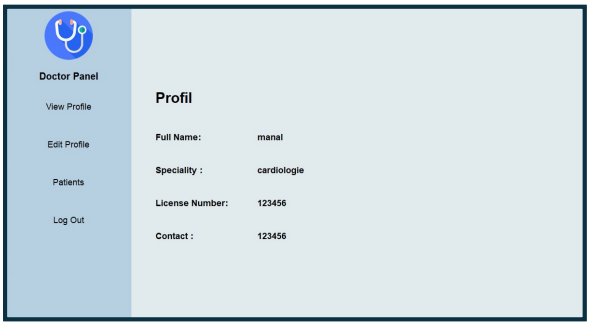
5.1 Error Handling



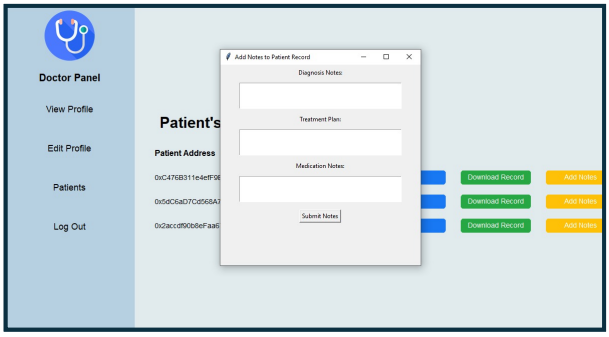


5.1 Doctor Panel



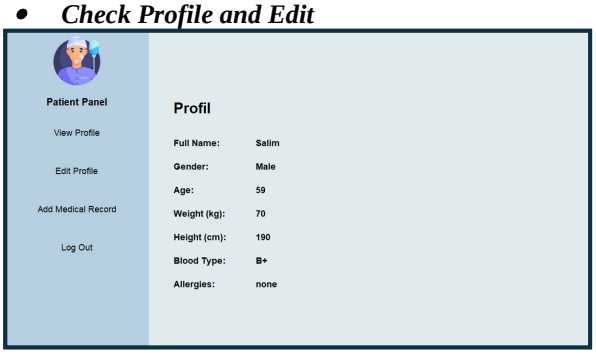


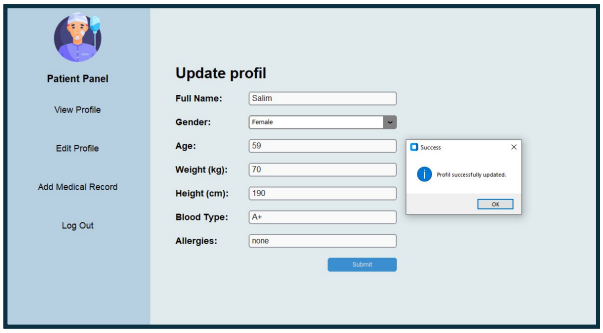
- ***Add Note***



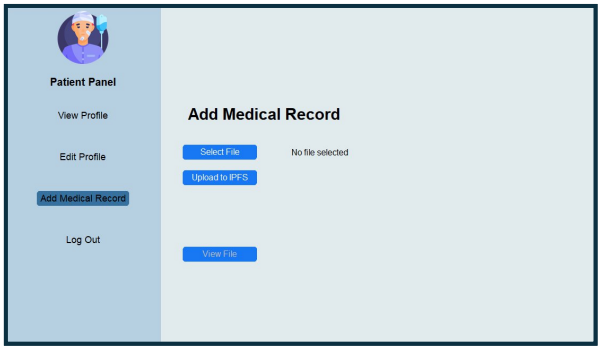
5.1 Patient Panel

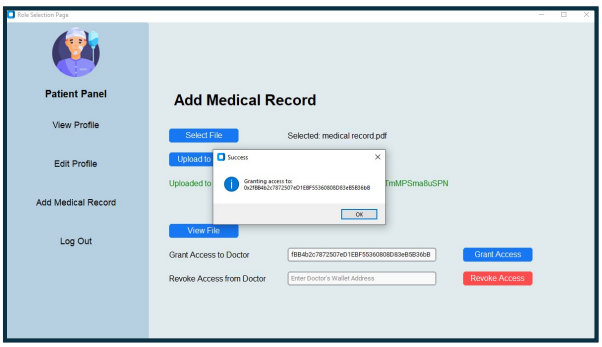
- ***Check Profile and Edit***

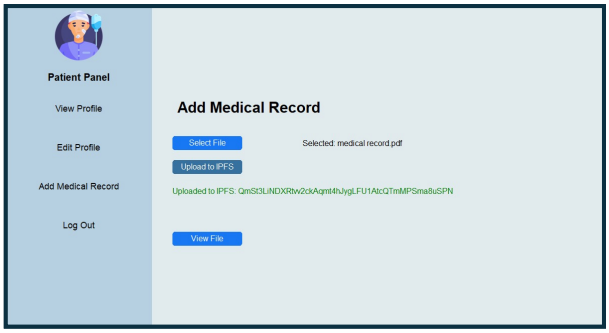




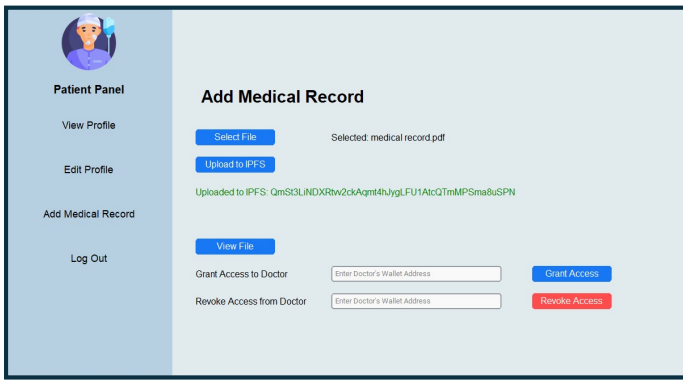
- ***Add Medical Record***



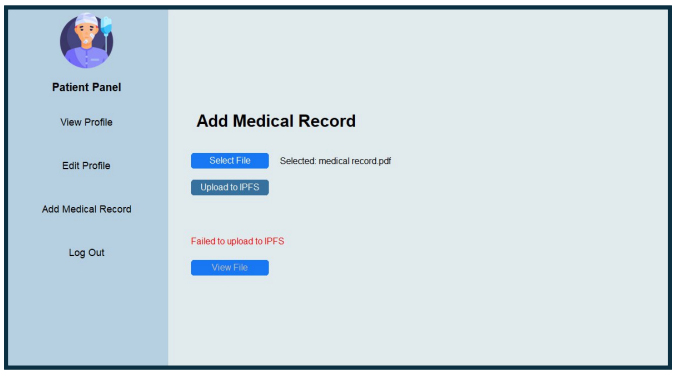




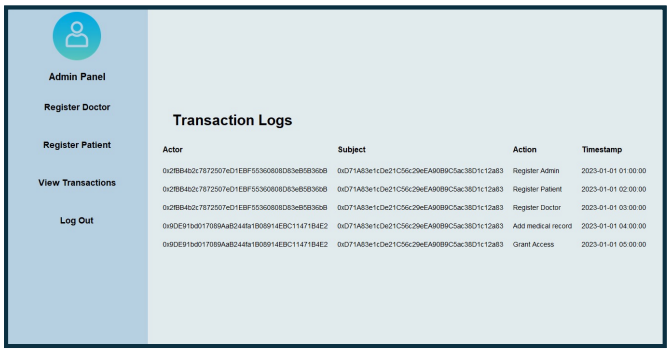
- ***Grant Access to Doctor for Medical Record***



- ***Error Handling***



5.1 Admin Panel



5.2 Performance Analysis

- Transaction times and IPFS upload efficiency are evaluated, demonstrating the system's capability to handle medical records securely and efficiently.

5.3 Security Evaluation

- Blockchain immutability and IPFS storage ensure robust protection against unauthorized data access and tampering.

6. DISCUSSION

6.1 Challenges

- Integrating IPFS with Ethereum posed initial difficulties in handling large medical files.
- Managing blockchain gas fees required optimization for resource efficiency.
- **MetaMask Integration Limitation:** While MetaMask is a common tool for validating blockchain transactions on web platforms, this project was designed for a desktop-based use case. The lack of native MetaMask support in Python necessitated manual management of private keys and signatures, increasing the complexity of transaction validation.

6.2 Future Enhancements

- Exploring the use of zero-knowledge proofs (zk-SNARKs) for enhanced privacy.
- Expanding interoperability with other blockchain platforms.
- Developing a MetaMask plugin or similar integration for desktop environments to streamline transaction validation.

7 CONCLUSION

This project demonstrates the feasibility of using blockchain and IPFS for secure medical record management. By ensuring confidentiality, integrity, and traceability, the system addresses the critical challenges of traditional healthcare data systems. The Python-based frontend enhances usability, making the solution accessible to a broad audience.

8 REFERENCES

[1] IBM/Medical-Blockchain https://github.com/IBM/Medical-Blockchain?utm_source=chatgpt.com

[2] "Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Centric Solutions" https://www.researchgate.net/publication/320147972_Blockchain_Technology_for_Healthcare

[3] MedRec: Blockchain for Medical Records

[4] Blockchain and IPFS healthcare application. https://www.patientory.com/

[5] Ethereum Foundation, "Ethereum Documentation", Available at: https://ethereum.org.

[6] IPFS Team, "IPFS Documentation", 2024. Available at: https://docs.ipfs.io.

[7] Web3 Foundation, "Web3.py: Python Ethereum Library", 2024. Available at: https://web3py.readthedocs.io.
[8] Reitwiessner, C., "zk-SNARKs for Blockchain Privacy", Ethereum Blog, 2016.

[9] Consensys Diligence, "Smart Contract Security Best Practices", Available at: https://consensys.io.

[10] ArrjunPradeep/Medic-Chain https://github.com/ArrjunPradeep/Medic-Chain?utm_source=chatgpt.com

[11] jayateertha043/Decentralized-Medical-Records https://github.com/jayateertha043/Decentralized-Medical-Records?utm_source=chatgpt.com

[12] SuyashMore/SwasthyaChain https://github.com/SuyashMore/SwasthyaChain?utm_source=chatgpt.com