



Department of Informatique,
Faculty of Sciences and Techniques,
Université Abdelmalek Essaâdi. Tanger, Morocco



End of module project: medical records using blockchain

Chibani Fahd, Samady Ahmad, Fakhre-eddine Mohamed Amine

December 16, 2024

I. Introduction:

Medical data protection is crucial in today's healthcare system. Traditional medical record systems often have significant security vulnerabilities, which puts patient information at risk of unauthorized access or data breaches. Blockchain technology offers an innovative solution to these challenges by providing a more secure and transparent way of managing medical records.

Our project develops a digital medical record system using blockchain technology. By leveraging Ethereum blockchain and smart contracts, we create a decentralized application that ensures patient data remains confidential, tamper-proof, and easily accessible to authorized users. The application uses Python with a user-friendly interface, making it both secure and simple to use.

The key innovation is the use of blockchain to solve critical healthcare data management problems. This approach guarantees that medical records are protected, traceable, and can be shared safely between healthcare professionals while maintaining patient privacy and data integrity.

II. Objectifs of Our project:

Medical Data Security:

The primary goal is to create an impenetrable shield for medical records using blockchain technology. Unlike traditional systems vulnerable to hacking and unauthorized modifications, the blockchain ensures that once data is recorded, it cannot be altered or deleted. Each medical record becomes a permanent, secure, and verifiable entry that protects patient information from potential tampering or unauthorized changes.

Privacy Protection:

Patient confidentiality is paramount in healthcare, and our system employs robust cryptographic techniques to ensure the security of medical data. By encrypting sensitive information, access is restricted exclusively to authorized personnel. Sophisticated encryption and access control mechanisms provide multiple layers of security, safeguarding individual patient details while enabling necessary medical professionals to access critical health information when required. This approach balances privacy with the accessibility essential for effective medical care.

Transparency and Tracking:

Every interaction with a medical record is made traceable and transparent through comprehensive logging and immutable tracking of all actions performed on medical files. Key features include a detailed history of data interactions, an unalterable record of access and modifications, and full auditability of every activity. This approach ensures unprecedented accountability and fosters trust in the secure and responsible management of medical records.

Decentralization:

By moving away from centralized database systems, we mitigate critical security challenges by eliminating single points of failure, reducing the risk of large-scale data breaches, and distributing medical records across multiple blockchain network nodes. This decentralized approach ensures that no single entity has complete control over the entire medical record database, significantly enhancing the system's overall security and resilience.

Interoperability:

Our blockchain solution will tackle the persistent challenge of fragmented healthcare information systems by:

- Creating a standardized, secure method for medical data exchange

- Enabling seamless communication between healthcare providers
- Maintaining strict privacy and security protocols
- Facilitating efficient sharing of patient information across different healthcare institutions

III. Solidity smart contracts:

Our blockchain medical record system uses four smart contracts to transform healthcare data management. By implementing secure, decentralized technology, we offer patients and doctors a transparent, tamper-resistant platform that guarantees data privacy, manages access control, and preserves a comprehensive, verifiable history of all medical information exchanges.

1. Patient Contract:

The **PatientContract** is a smart contract written in Solidity designed to manage patient medical records on the Ethereum blockchain in a decentralized manner. The contract includes functionalities for patient registration, granting and revoking doctor access and storing medical files, all while ensuring secure and transparent handling of data.

Key Features and Functionalities:

Key features of the contract include patient registration, access control, medical file management and data retrieval. Patients can register by providing personal information such as full name, date of birth, phone number, and gender. Once registered, patients have the ability to grant or revoke access to their medical records for specific doctors. Medical files, stored as IPFS hashes, can be added or deleted, and patients can add files to their records. Authorized doctors can retrieve medical files, while patients have the ability to view their own records.

Contract Structure:

The contract is built using Solidity version 0.8.0, incorporating the OpenZeppelin libraries **Counters** for managing counters and **ReentrancyGuard** to prevent reentrancy attacks. It defines several data structures, such as the **Patient** structure, which holds the patient's address, personal details, and registration status; the **DoctorAccess** structure, which tracks a doctor's access to patient records; and the **MedicalFile** structure, which holds the IPFS hash and name of medical files.

Mappings and Counters:

The contract uses mappings for efficient data organization:

- `mapping(address => Patient) patients`: Maps patient addresses to their corresponding information.
- `mapping(address => mapping(address => DoctorAccess)) patientDoctorAccess`: Tracks access permissions between patients and doctors.
- `mapping(address => MedicalFile[]) patientMedicalFiles`: Stores medical files for each patient.
- `mapping(address => string[]) patientComments`: Stores comments associated with a patient.

The `patientCounter` is a `Counters.Counter` instance used to track the total number of registered patients.

Events:

Several events are emitted to notify the blockchain about key actions, such as when a patient registers, when a doctor is granted or revoked access, when a medical file is added.

Core Functions:

The contract includes the following key functionalities:

* Patient Registration:

- **registerPatient**: Allows a user to register as a patient by providing their personal information.
- **isPatientRegistered**: Checks if a patient is registered using their Ethereum address.

* Doctor Access Management:

- **grantDoctorAccess**: Enables a patient to grant a doctor access to their records.
- **revokeDoctorAccess**: Allows a patient to revoke a doctor's access.
- **checkDoctorAccess**: Verifies if a doctor has access to a patient's records.

* Medical File Management:

- **addMedicalFile**: Adds a medical file to a patient's record, identified by its IPFS hash.
- **deleteMedicalFile**: Allows a patient to remove a medical file from their records.
- **getMedicalFiles**: Retrieves all medical files associated with a patient for authorized doctors.

* Information Retrieval:

- **getPatientInfo**: Retrieves personal information of the calling patient.
- **getOwnMedicalFiles**: Retrieves medical files associated with the calling patient.

* **GetTotalPatients**: Returns the total number of registered patients.

Security Measures:

The contract includes robust security measures, utilizing the **ReentrancyGuard** modifier to protect against reentrancy attacks. Access to sensitive data is controlled through mappings and conditional checks.

2. Doctor Contract:

The **DoctorContract** provides a range of features including doctor registration, uploading of medical files, retrieval of patient medical records, and deletion of a doctor's own files. It ensures that only authorized and registered doctors can perform specific actions. The contract interacts with the **PatientContract** to verify permissions and store patient-related medical files. Doctors are able to maintain their own record of uploaded medical files while providing services to patients.

Contract Structure:

The contract utilizes Solidity version 0.8.0 and incorporates the **ReentrancyGuard** library from OpenZeppelin to protect against reentrancy attacks. It defines multiple structures and mappings to organize and manage data, such as the **Doctor** structure and the **MedicalFile** structure.

Data Structures:

* **Doctor**: Represents a doctor and includes the following attributes:

- **address doctorAddress**: Ethereum address of the doctor.
- **string fullName**: Full name of the doctor.
- **string dateOfBirth**: Date of birth of the doctor.
- **string phoneNumber**: Phone number of the doctor.
- **string gender**: Gender of the doctor.

- `string speciality`: Area of medical specialization.
 - `string licenseNumber`: License number of the doctor.
 - `bool isRegistered`: Indicates if the doctor is registered.
- * **MedicalFile**: Represents a medical file uploaded by a doctor with the following attributes:
- `string ipfsHash`: Hash of the file stored on IPFS.
 - `string name`: Name of the medical file.

Mappings and Counters:

The contract uses mappings to associate doctors with their data and uploaded files:

- `mapping(address => Doctor) doctors`: Maps doctor addresses to their details.
- `mapping(address => MedicalFile[]) doctorMedicalFiles`: Maps a doctor's address to their list of uploaded medical files.

A counter, `doctorCounter`, is used to track the total number of registered doctors.

Events:

The contract emits events to notify about key actions:

- **DoctorRegistered**: Emitted when a doctor successfully registers.
- **MedicalFileUploaded**: Emitted when a doctor uploads a medical file for a patient.

Core Functions:

The contract defines the following core functions:

- **registerDoctor**: Allows a doctor to register by providing personal details such as name, date of birth, phone number, gender, specialization, and license number. Doctors can only register once.
- **uploadMedicalFile**: Enables a registered doctor to upload a medical file for a patient, provided they have access to the patient's records via the **PatientContract**.
- **getDoctorInfo**: Retrieves detailed information about a doctor, including their full name, date of birth, phone number, gender, specialization, and license number.
- **getPatientMedicalFiles**: Allows a doctor to retrieve the medical files of a patient, subject to access permissions managed by the **PatientContract**.
- **deleteOwnMedicalFile**: Allows a doctor to delete a specific medical file they previously uploaded, identified by its IPFS hash.
- **getDoctorMedicalFiles**: Retrieves the list of medical files uploaded by the requesting doctor.
- **getTotalDoctors**: Returns the total number of registered doctors in the system.

Security Measures:

The contract ensures robust security through the following measures:

- **Access Control**: Verifies that only registered doctors can perform specific actions.
- **ReentrancyGuard**: Protects functions from reentrancy attacks using the `nonReentrant` modifier.
- **Permission Checks**: Interacts with the **PatientContract** to ensure that a doctor has explicit permission before accessing a patient's records.

3. Audit Contract:

The **AuditContract** is a Solidity smart contract designed to provide a transparent and accountable system for tracking actions and events within a blockchain-based healthcare application. By logging significant activities and granting controlled access to auditors, the contract ensures an immutable and traceable record of all interactions.

Key Features and Functionalities

The **AuditContract** offers the following capabilities:

- Comprehensive event logging, including patient and doctor registrations, access grants or revocations, and file uploads.
- Controlled access for auditors to view and retrieve the audit trail.
- Transparent and immutable storage of audit logs, enhancing accountability.

Contract Structure

The contract leverages Solidity version 0.8.0 and incorporates OpenZeppelin's **Counters** utility. Below are its primary components and mechanisms.

Data Structures

The **AuditContract** employs a structured approach to organize audit data:

- * **AuditEvent**: Represents a logged event, storing:
 - `uint256 timestamp`: The time when the event occurred.
 - `address actor`: The Ethereum address of the entity performing the action.
 - `string eventType`: A label describing the type of event (e.g., `PATIENT_REGISTRATION` or `MEDICAL_FILE_UPLOAD`).
 - `string description`: A detailed description of the event.
- * **Auditors**: Maintains a list of authorized auditors who can access the audit logs.

Mappings and Counters

The contract uses mappings and counters for efficient data management:

- `mapping(uint256 => AuditEvent) auditTrail`: Maps unique event IDs to their corresponding audit events.
- `Counters.Counter eventCounter`: Tracks the total number of logged events.
- `mapping(address => bool) auditors`: Keeps track of authorized auditors.
- `address[] registeredAuditors`: Stores a list of all registered auditors.

Events

The contract emits an event for every logged action:

- **EventLogged**: Emitted when a new event is recorded, containing the event ID, actor, type, and description.

Core Functions

The contract includes the following key functionalities:

* Auditor Management:

- `addAuditor`: Allows an existing auditor to add a new auditor.
- `getAuditors`: Retrieves the list of all authorized auditors.
- `isAuditor`: Checks if the caller is an authorized auditor.

* Event Logging:

- `logPatientRegistration`: Logs the registration of a patient.
- `logDoctorRegistration`: Logs the registration of a doctor.
- `logDoctorAccessGrant` and `logDoctorAccessRevoke`: Logs the granting or revocation of access to doctors.
- `logDoctorMedicalFileUpload` and `logPatientMedicalFileUpload`: Logs the upload of medical files by doctors or patients.

* Audit Trail Management:

- `getAuditEvent`: Retrieves the details of a specific event by its ID.
- `getFullAuditTrail`: Returns a complete list of all logged events.
- `getTotalEvents`: Provides the total number of logged events.

Security and Transparency

The `AuditContract` enhances system integrity and trust by:

- Enforcing strict access control for auditors to view sensitive logs.
- Maintaining an immutable history of all significant actions.
- Providing detailed and traceable logs to ensure transparency and accountability.

4. RoleContract contract:

The `RoleContract` is responsible for handling the authentication process within the system. It ensures that each user whether a patient, doctor, or auditor has a unique address. This contract prevents the possibility of multiple users sharing the same Ethereum address. By verifying and maintaining the uniqueness of each address, the contract ensures proper identification and prevents conflicts in user roles, allowing for secure and distinct access for each participant in the system.

Scalability:

To ensure scalability, the contract leverages IPFS for decentralized storage of medical files, meaning that only essential data like metadata and access control information is stored on the blockchain, reducing storage load and enhancing scalability.

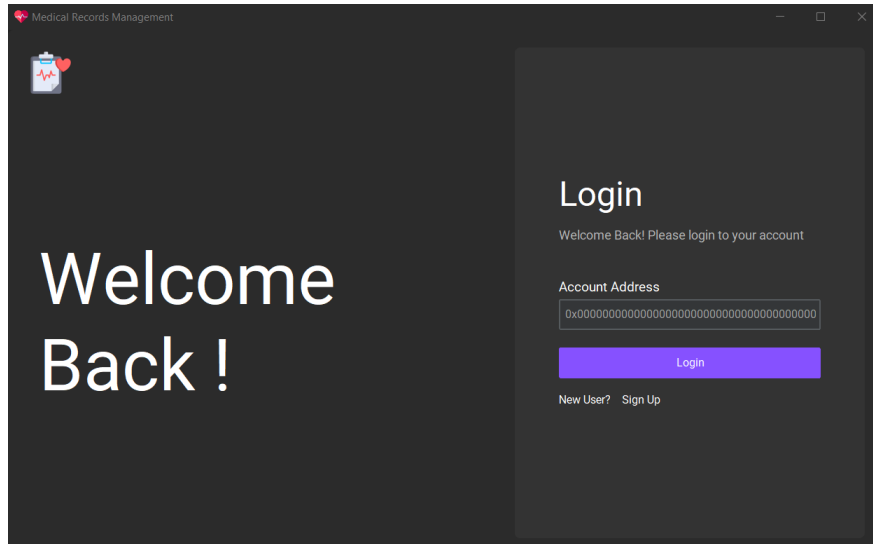
IV. Application:

To interact with our Solidity contract, we created a desktop application using the Python libraries `customtkinter` for the user interface and `web3` for blockchain communication. The application includes five main frames: a Registration Frame, where doctors and patients can create their accounts; a Login Frame, which allows users to log in and access their accounts; a Patient Frame, where patients can view their information and add doctors; a Doctor Frame, which enables doctors to view their own information and access details about their patients; and an Audit Frame, where events and transactions on the blockchain can be monitored.

Login Frame:

The Login Frame allows users whether patients, doctors, or auditors to log into their accounts by providing their account address. To confirm ownership of the account, users must provide a signature for authentication. This ensures secure access to their respective accounts. Additionally, if a user is not yet registered, they can navigate to the Registration Page from this frame to create a new account.

- Login frame:

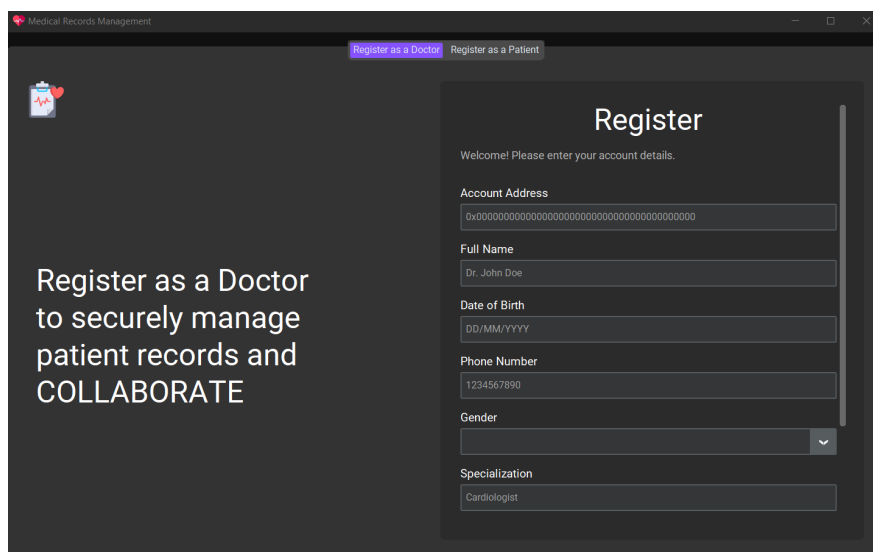


The screenshot shows a web application window titled "Medical Records Management". On the left, there is a large white text area that says "Welcome Back !". On the right, there is a dark gray login form. The form has a title "Login" and a subtitle "Welcome Back! Please login to your account". It contains a text input field labeled "Account Address" with a placeholder "0x00". Below the input field is a blue "Login" button. At the bottom of the form, there are two links: "New User?" and "Sign Up".

Registration Frame:

The Registration Frame consists of two subframes, the patient and doctor registration. In the Patient Registration subframe, patients provide their personal information such as their name, address, date of birth, and contact details. In the Doctor Registration subframe, doctors input their information, including their address, medical specialty, and license number. Once the registration process is successfully completed, the user is automatically redirected to the Login Frame to access their account.

- Doctor registration sub-frame:



The screenshot shows a web application window titled "Medical Records Management". At the top, there are two tabs: "Register as a Doctor" (which is active) and "Register as a Patient". On the left, there is a large white text area that says "Register as a Doctor to securely manage patient records and COLLABORATE". On the right, there is a dark gray registration form. The form has a title "Register" and a subtitle "Welcome! Please enter your account details.". It contains several input fields: "Account Address" (placeholder: "0x00"), "Full Name" (placeholder: "Dr. John Doe"), "Date of Birth" (placeholder: "DD/MM/YYYY"), "Phone Number" (placeholder: "1234567890"), "Gender" (a dropdown menu), and "Specialization" (placeholder: "Cardiologist").

- Patient registration sub-frame:

The screenshot shows the 'Medical Records Management' application window. At the top, there are two tabs: 'Register as a Doctor' and 'Register as a Patient', with the latter being selected. On the left side, there is a dark grey panel with a white medical icon and the text: 'Are you a Patient? Register to share your medical records SECURELY'. The main area is a light grey 'Register' form. It includes a welcome message: 'Welcome! Please enter your account details.' and several input fields: 'Account Address' (with a placeholder '0x00'), 'Full Name' (with 'John Doe'), 'Date of Birth' (with 'DD/MM/YYYY'), 'Phone Number' (with '1234567890'), and 'Gender' (a dropdown menu). A blue 'Register' button is at the bottom of the form. Below the button, there is a link: 'Already have an Account? Login'.

Patient Frames:

The Patient Frames allows patients to manage their personal and medical information. Patients can view the personal details they provided during the registration process and upload additional information, such as medical files. They also have the option to delete existing medical files if needed. Furthermore, patients can grant access to doctors using their Ethereum address, allowing doctors to view the patient's information and upload medical files on their behalf. Patients can download these uploaded medical files for their records. Additionally, patients have the ability to revoke access from doctors at any time, ensuring control over who can view or manage their medical data.

- Patient profile sub-frame:

This Sub-Frame allows patients to view their personal information securely. This includes details such as their name, address, date of birth, contact information, and gender, which they provided during the registration process.

The screenshot shows the 'Medical Records Management' application window. At the top, there are three tabs: 'Profile', 'Grant/Revoke Access', and 'Medical Records', with 'Profile' being selected. The main area is a light grey 'Profile' page. It displays the following information: 'Logged in as: 0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199', 'Name: Ahmed Samady', 'Date of Birth: 02/11/2002', 'Phone Number: 0217498424', and 'Gender: Male'. At the bottom center, there is a blue 'Logout' button.

- Grant and revoke access sub-frame:

This sub-Frame enables patients to manage access permissions for their personal information. Patients can grant access to doctors, allowing them to view their personal details and upload additional information. Additionally, patients retain full control and can revoke this access at any time, ensuring their privacy and security.

The screenshot shows a web application window titled "Medical Records Management". It has three tabs: "Profile", "Grant/Revoke Access" (which is active), and "Medical Records". The main heading is "Grant/Revoke Access". Below this, there are two columns. The left column is titled "Grant Access" and contains a text input field labeled "Doctor's Address" and a blue button labeled "Grant Access". The right column is titled "Revoke Access" and contains a text input field labeled "Doctor's Address" and a blue button labeled "Revoke Access".

- Patient medical records sub-frame:

This sub-Frame allows patients to manage their medical files efficiently. Patients can upload their own medical files as well as download files shared by their doctors. Additionally, they have the ability to delete any files at any time. This sub-frame displays each file's name along with its corresponding IPFS hash for easy identification.

The screenshot shows a web application window titled "Medical Records Management". It has three tabs: "Profile", "Grant/Revoke Access", and "Medical Records" (which is active). The main heading is "Medical Records". Below this, there is a table with two columns: "IPFS Hash" and "File Name". The table contains four rows of data. Below the table, there are three buttons: "Download Selected" (green), "Upload New File" (blue), and "Delete Selected" (red).

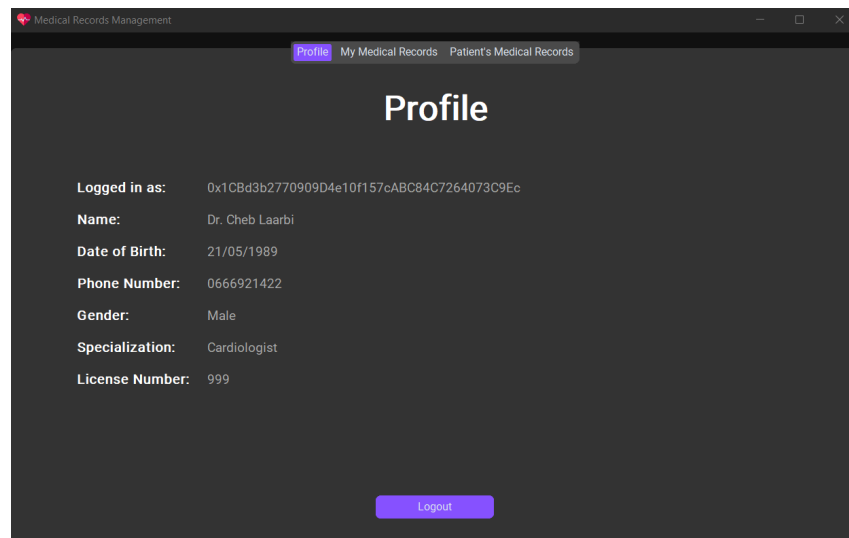
IPFS Hash	File Name
QmZqM9rT2vKtEpxLoMDUd0vEWKpg7xJLbXGneeZ3oZe	02- Mini project Topics.xlsx
QmTVuW825o13GmXXZKXs22yEtqP8BvDdFF3k5EPYUuJHo	ats-friendly-technical-resume.pdf
QmeDut5qtHfu5wKhTh2khlUpqJJDLgkxPFS17PvmqM4dh	Alkon, Eminem - Smack That.mp3
QmQCgRg8a8jRtA8cZ7gh1aficLDEZFGC96Nux4bbH1Qv	base-rover.pdf

Doctor Frames:

The Doctor Frames allows doctors to manage both their personal information and the medical information of patients who have granted them access. Doctors can view the personal details they provided during the registration process and access information of patients who have permitted them. Within this frame, doctors can upload medical files for these patients and view their existing medical records. Additionally, doctors have access to a personal library where they can store medical files of patients. This library allows doctors to save files they want to keep for future reference or to send later, ensuring efficient management of patient-related documents while maintaining privacy and control.

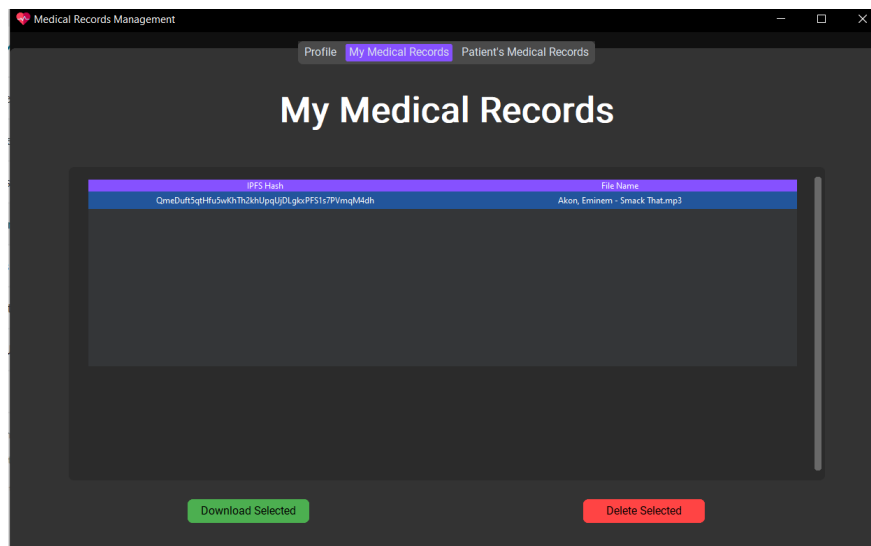
- Profile sub-frame:

Sub-Frame displays the doctor's personal information, including their name, specialty, and license number, providing a clear overview of their professional credentials.



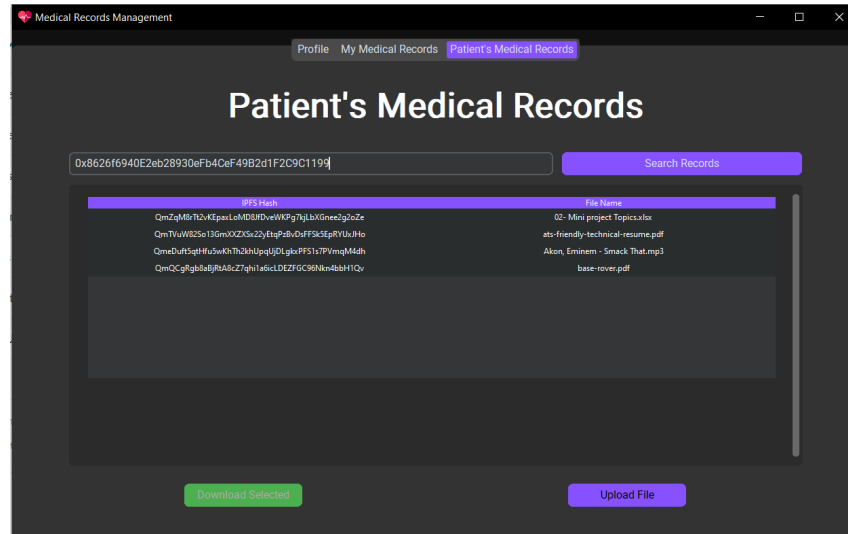
- Medicales Records library sub-frame:

This sub-Frame allows doctors to store medical records for future reference. This feature enables them to review and use the records as references when needed.



- Patient medical records:

This sub-Frame allows doctors to upload medical files directly to their patients' records and view the medical files shared by their patients. This feature facilitates seamless collaboration and exchange of critical medical information between doctors and patients.

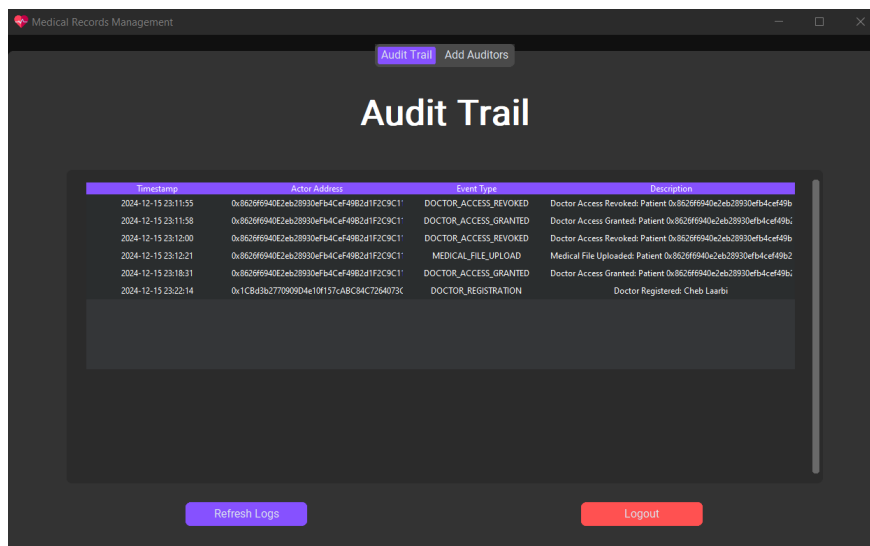


Audit Frame:

The Audit Frame provides a comprehensive overview of system activities and access management. It includes a table that displays all recorded events with their corresponding timestamps, offering a detailed log of actions such as file uploads, access grants, and revocations. Additionally, there is a section dedicated to managing access control, where permissions can be reviewed and granted. A specific area is also reserved for auditors who have exclusive access to this frame, enabling them to monitor and analyze all events and access-related activities. This ensures transparency, accountability, and efficient oversight of the entire system.

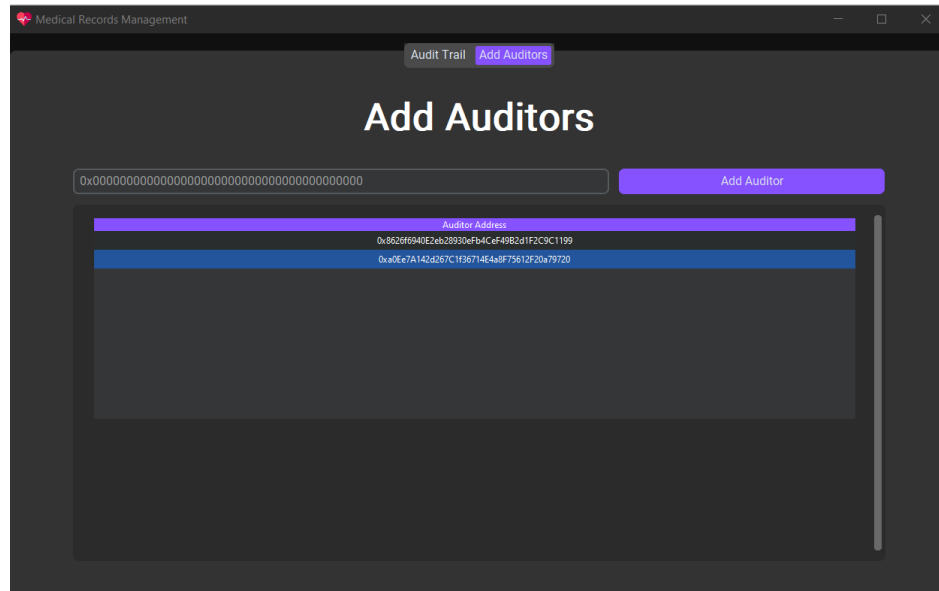
- Audit Trail sub-frame :

This sub-Frame allows auditors to track all events in the system, displayed with the timestamp, the actor (the individual who performed the event), and a description of the event. This feature provides full transparency and accountability, enabling auditors to verify and monitor all interactions with patient records and other important actions.



- Audit Auditors sub-frame:

This sub-Frame allows the system administrators to grant access to auditors who need to track events in the audit trail. Access is provided by adding the auditor's address, enabling them to view and monitor all recorded actions and events. This ensures that only authorized individuals can access sensitive audit data



Technologies used :

In this project, various technologies were employed to create a secure and decentralized medical records management system. The following technologies were used:

Remix IDE:

Remix IDE is an open-source web application for smart contract development. It provides a powerful interface for writing, compiling, testing, and deploying Solidity contracts directly on the Ethereum blockchain. Remix makes it easy to interact with the Ethereum Virtual Machine (EVM) and deploy contracts on both testnets and the main Ethereum network.



MetaMask:

MetaMask is a popular browser extension that acts as a bridge between the user and the Ethereum blockchain. It allows users to manage their Ethereum accounts, send transactions, and interact with decentralized applications (dApps) securely. It is used in this project for managing user wallets and for transaction signing.



MyCrypto:

MyCrypto is an open-source Ethereum wallet that allows users to manage their Ethereum accounts securely. It provides a way to check Ethereum account balances, transfer funds, and interact with smart contracts. MyCrypto is used to verify user accounts by providing a unique signature .



Hardhat:

Hardhat is a development environment for compiling, testing, and deploying Ethereum software. It helps developers work with the Ethereum blockchain in a local environment by providing test networks and easy access to test Ether for deploying and interacting with smart contracts. In this project, Hardhat is used to create and manage test Ethereum accounts and for running tests on the smart contracts.



Solidity:

Solidity is a contract-oriented programming language used for writing smart contracts on the Ethereum blockchain. It is the primary language for developing decentralized applications (dApps). In this project, Solidity is used to write the contracts responsible for managing patient and doctor interactions, access control, and medical file management.



IPFS for Storing Medical Records

InterPlanetary File System (IPFS) is a decentralized storage solution that allows users to store and retrieve files in a distributed manner. It is used in this project to securely store medical records, ensuring that files are immutable and accessible only to authorized users (doctors and patients).



CustomTkinter:

CustomTkinter is a Python library that extends Tkinter to create modern, visually appealing graphical user interfaces (GUIs). In this project, CustomTkinter is used to design the desktop application for patients and doctors to interact with the blockchain. The user interface provides a simple and intuitive way for users to register, view records, and interact with the Ethereum blockchain.



Web3:

Web3.js is a JavaScript library that allows developers to interact with the Ethereum blockchain. It is used in this project to connect the front-end user interface with the Ethereum blockchain, enabling users to send transactions, call smart contract functions, and interact with decentralized applications.



Conclusion:

This project presents a blockchain-based healthcare management system using Solidity smart contracts for patient and doctor record management, as well as audit tracking. The desktop application, built with Python libraries customtkinter for the user interface and web3 for blockchain communication, provides a user-friendly interface with five main frames: registration, login, patient, doctor, and audit. The system ensures secure, transparent interactions by encrypting medical data, granting controlled access to authorized users, and maintaining an immutable log of all activities. By decentralizing data storage and eliminating single points of failure, the solution enhances security and privacy while ensuring accountability in medical record management.