PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage

Bengaluru 560085

Department of Computer Science and Engineering

B. Tech. CSE - 6th Semester
Jan – May 2025

UE22CS342BA5–BLOCKCHAIN

PROJECT REPORT

On

Personalized Medical Vault

Submitted by : Team #**BitHealth**

Mujaseem D: PES1UG22CS363

N Swetha: PES1UG22CS368

Likhit Avinash V: PES1UG22CS304

Kripa S Rai: PES1UG22CS291

Class of Prof. Indu R

Table of Contents

1. INTRODUCTION

The rise of blockchain technology has introduced a paradigm shift in the way data can be stored and managed. In the healthcare sector, where data sensitivity and privacy are paramount, blockchain offers significant advantages in terms of security, immutability, and transparency. This project, titled Personalised Medical Vault, is a decentralized application that leverages Ethereum blockchain and smart contracts to manage and store medical records. It aims to replace traditional centralized health record systems, which are often prone to data breaches and lack interoperability, with a secure, tamper-proof, and accessible platform for both patients and healthcare providers.

A Personalized Medical Vault is built on blockchain and operates without a central authority, ensuring transparency and data integrity.

2. PROBLEM STATEMENT

In today's healthcare ecosystem, patients often lack control over their own medical data. Medical records are typically stored in centralized hospital or clinic databases, making access inefficient and dependent on institutional policies. Patients cannot easily view, share, or revoke access to their records, and this fragmented approach often leads to delays, repeated tests, or even critical information gaps during treatment.

The absence of a unified, secure, and patient-controlled system limits transparency and makes it difficult to manage personal health data efficiently.

This project addresses the need for a decentralized, user-centric solution by developing a **Personalized Medical Data Vault**. This platform empowers patients to:

- Upload their medical records securely using IPFS.

- Manage access to these records through a blockchain-based smart contract.
- Grant or revoke access to doctors in real-time without relying on third parties.

Doctors, in turn, get a dedicated dashboard to view only the patients who have explicitly shared records with them.

By decentralizing data control and giving ownership back to the patient, the platform ensures privacy, transparency, and quick access — essential factors for modern, efficient, and ethical healthcare systems.

3. TECHNOLOGIES USED

Frontend : React.js, JavaScript, Web3.js
Blockchain : Ethereum, Solidity (v0.8.x)
Smart Contract Framework : Truffle (v5.9.0)
Local Blockchain : Ganache CLI
Browser Wallet : MetaMask
Development Tools : Node.js, npm
File system: IPFS

4. SYSTEM OVERVIEW

This system features two primary user roles : Patient and Doctor.

4.1 Functional Workflow:

1. Patient Registration: Patients connect their MetaMask wallet and register.
2. Record Upload: Patients can upload medical records via the React interface.
3. Access Granting: Patients can grant/revoke access to their records for doctors.
4..Doctor Access: Doctors can view patient records only after access is granted.
5. Smart Contracts: Handle access control and record management on-chain.

4.2 Actors and Interfaces:

- Patients: Full ownership of their data, upload, and manage access.
- Doctors: View records with permission, no upload/edit rights.
- MetaMask: Used for transaction signing and wallet connectivity.
- Ganache: Local test Ethereum blockchain.
- React UI: User-friendly frontend built with Web3.js for blockchain interaction.

5. BLOCKCHAIN IMPLEMENTATION DETAILS

5.1 Frontend(client/)

The React-based frontend ensures a seamless experience:

- User registration with role selection
- File upload and data permission management
- Real-time blockchain data rendering using Web3.js

React hooks like useState and useEffect manage app state and interactions. MetaMask integration secures user identity and transaction approvals.

5.2.Smart Contracts(medrecords-dapp/)

Smart contracts written in Solidity contain the core logic. The main contract MedicalRecord.sol defines:

- Data structures for storing records (patient ID, doctor ID, diagnosis, prescriptions, timestamps)
- Core functions:
    - addRecord: For adding new records
    - viewRecord(): For authorized viewing
    - grantAccess() and revokeAccess()

Access is enforced using custom modifiers and require statements. Events are emitted to notify the frontend of changes. Contracts are deployed via Truffle and tested on Ganache.

Smart Contract – MedicalRecords.sol

The contract is developed in **Solidity v0.8.19** and licensed under the **MIT License**.

```
address public owner;
```

Owner holds the address of the contract deployer and can be used for administrative access or audit purposes.

```
enum Role { None, Patient, Doctor }
```

An **enum** named **Role** defines possible roles:

- **None**: Default value before registration.
- **Patient**: User who uploads and controls medical records.
- **Doctor**: User who may request access to patient records.

Record Structure

```
struct Record {
   string ipfsHash;
   address uploader;
}
```

Defines the structure of a medical record:

- **ipfsHash**: A unique identifier for the file stored on IPFS.
- **uploader**: The address of the user who uploaded the record.

```
mapping(address => Record[]) private records;
mapping(address => Role) public userRoles;
mapping(address => mapping(address => bool)) public accessPermissions;
```

records: Maps a user's address to their array of uploaded medical records.
userRoles: Stores the role (Patient/Doctor) for each address.
accessPermissions: Nested mapping to manage doctor access permissions per patient.

Events

These events emit logs that frontend apps can use.

```
  event RecordUploaded(...);

  event UserRegistered(...);

  event AccessGranted(...);

  event AccessRevoked(...);
```

UserRegistered: Emitted when a user registers as a Patient or Doctor.

RecordUploaded: Emitted when a Patient uploads a medical record.

AccessGranted: Emitted when a Patient grants a Doctor access to their records.

AccessRevoked: Emitted when a Patient revokes access previously granted to a Doctor.

```
  constructor() {

      owner = msg.sender;

  }
```

Constructor: Sets the contract deployer as the owner.

Registering Users

```
    function registerUser(uint8 role) external
```

Allows users to register once as either a Patient or Doctor .

Rejects re-registration and invalid role inputs.

Emits the **UserRegistered** event on success.

Uploading Records

function uploadRecord(string calldata ipfsHash) external

Allows only registered Patients to upload records.

Pushes a new **Record** to the caller's record list.

Ensures that patients have control over their health data.

Emits the **RecordUploaded** event.

Viewing Records

function getRecords(address user) external view returns (Record[] memory)

Allows access to records only if:

The caller is the owner (admin).

The caller is the user themselves.

The caller is a Doctor with permission from the user.

Prevents unauthorized data access.

Granting Access

function grantAccess(address doctor) external

Let's a Patient grant access to their records to a specific Doctor.
Verifies that the target address is indeed a Doctor.
Updates the **accessPermissions** mapping accordingly.
Emits the **AccessGranted** event.

Revoking Access

function revokeAccess(address doctor) external

Let's a Patient revoke access previously given to a Doctor.

Resets the permission to false.

Emits the **AccessRevoked** event for tracking.

6. DEPLOYMENT

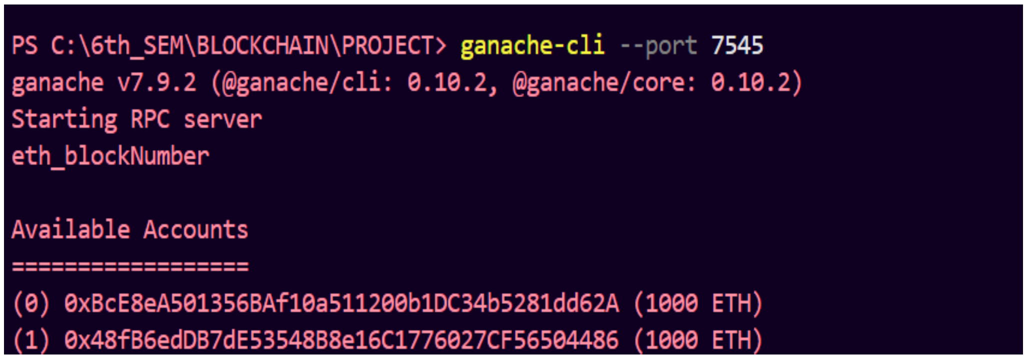
6.1 Prerequisites

- Install Node.js and npm
- Install Truffle globally:
  npm install -g truffle
- Install and launch Ganache (GUI or CLI)
- Install MetaMask in your browser

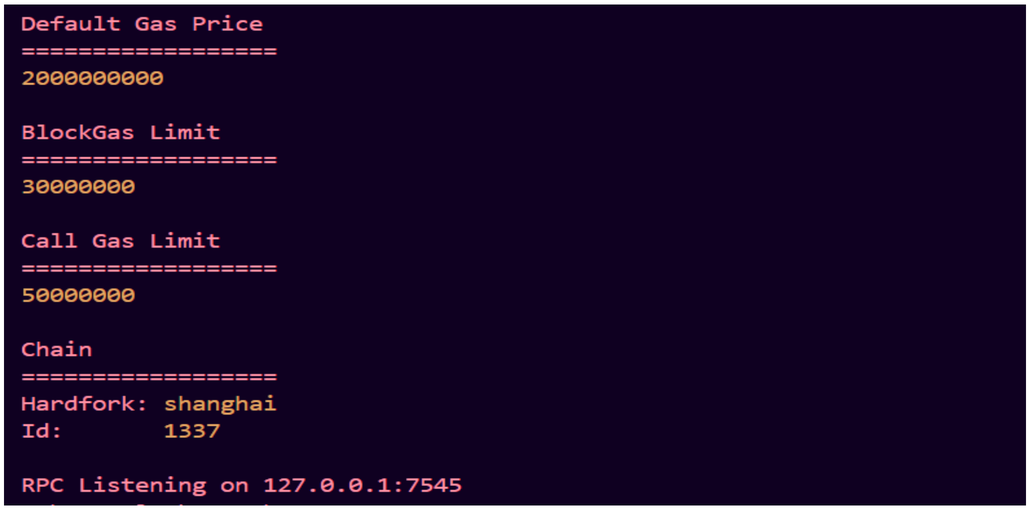
6.2 Setup Steps

- **Starting the Ganache CLI server**

ganache-cli –port 7545



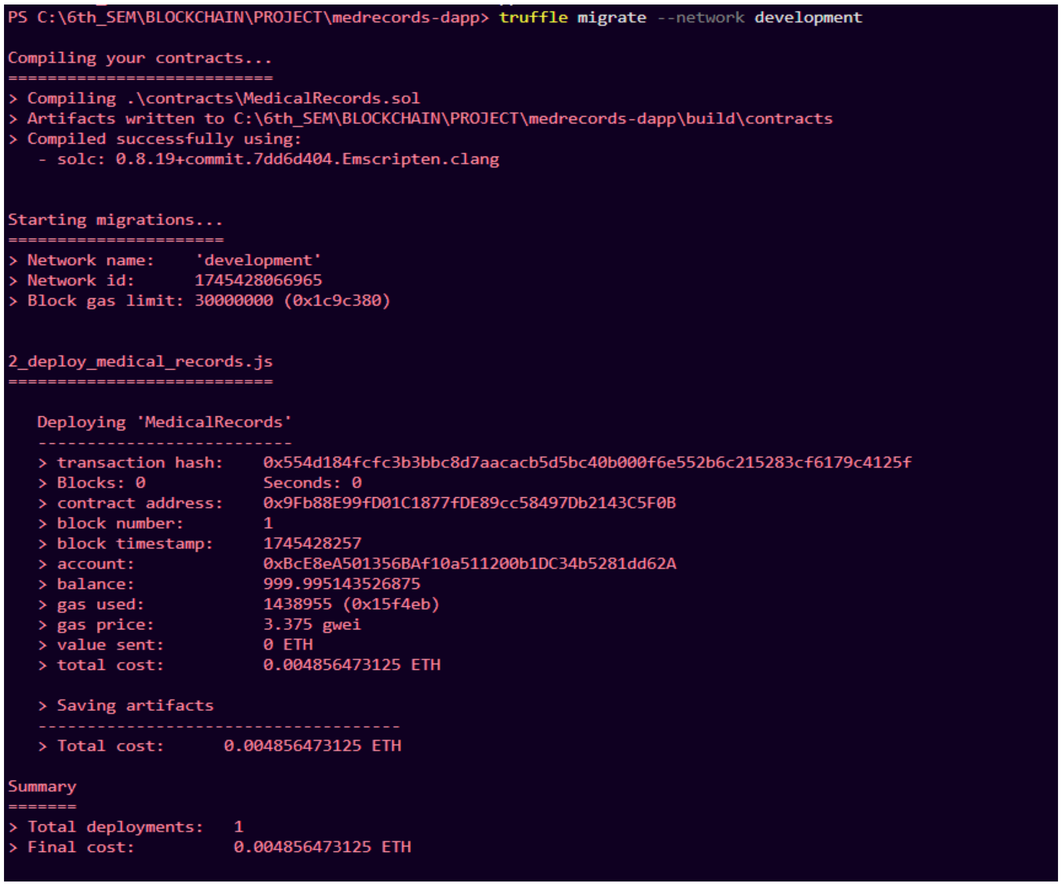
```
PS C:\6th_SEM\BLOCKCHAIN\PROJECT> ganache-cli --port 7545
ganache v7.9.2 (@ganache/cli: 0.10.2, @ganache/core: 0.10.2)
Starting RPC server
eth_blockNumber

Available Accounts
==================
(0) 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A (1000 ETH)
(1) 0x48fB6edDB7dE53548B8e16C1776027CF56504486 (1000 ETH)
```



```
Default Gas Price
==================
2000000000

BlockGas Limit
==================
30000000

Call Gas Limit
==================
50000000

Chain
==================
Hardfork: shanghai
Id:       1337

RPC Listening on 127.0.0.1:7545
```

Truffle -migration:

truffle migrate –network development



```
PS C:\6th_SEM\BLOCKCHAIN\PROJECT\medrecords-dapp> truffle migrate --network development

Compiling your contracts...
===========================
> Compiling .\contracts\MedicalRecords.sol
> Artifacts written to C:\6th_SEM\BLOCKCHAIN\PROJECT\medrecords-dapp\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      1745428066965
> Block gas limit: 30000000 (0x1c9c380)


2_deploy_medical_records.js
===========================

   Deploying 'MedicalRecords'
   --------------------------
   > transaction hash:    0x554d184fcfc3b3bbc8d7aacacb5d5bc40b000f6e552b6c215283cf6179c4125f
   > Blocks: 0            Seconds: 0
   > contract address:    0x9Fb88E99fD01C1877fDE89cc58497Db2143C5F0B
   > block number:        1
   > block timestamp:     1745428257
   > account:             0xBcE8eA501356BAf10a511200b1DC34b5281dd62A
   > balance:             999.995143526875
   > gas used:            1438955 (0x15f4eb)
   > gas price:           3.375 gwei
   > value sent:          0 ETH
   > total cost:          0.004856473125 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:      0.004856473125 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.004856473125 ETH
```

- **MetaMask Setup (Wallet Connection)**

  Ethereum Blockchain is being Used.

  import an Ethereum wallet with the secret key generated by Ganache CLI.

- **Pinata Setup (IPFS File Hosting)**

  Pinata makes it easy to upload files to IPFS and get a permanent link (CID).

- **IPFS File Upload (Frontend Integration with Pinata)**

  Using the API Key And Secret Key to establish the connection.

  Dependencies to install:

  npm install axios

- **Storing the IPFS Hash on Blockchain (Smart Contract + MetaMask)**

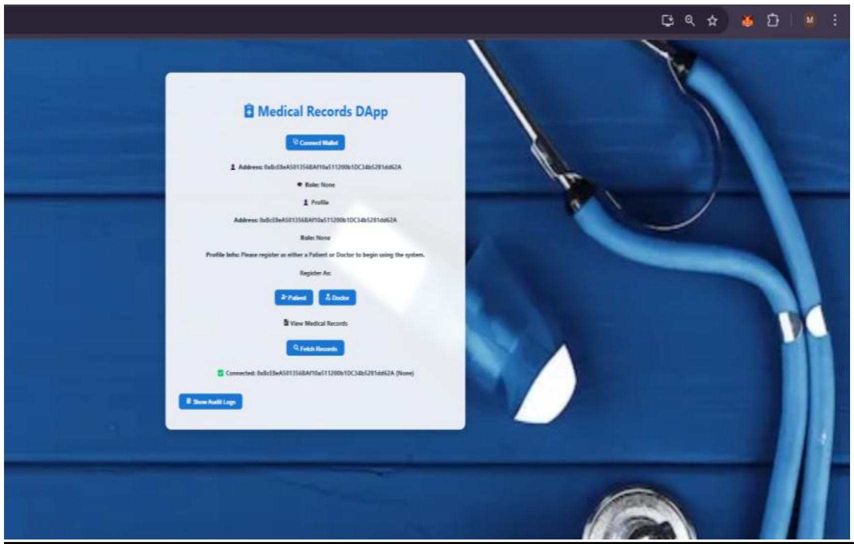
  Connect MetaMask in React

  Interaction with the deployed smart contract

- **Testing the Full Flow**

  Start the React app: npm start

  Localhost:3000



Medical Records DApp

Connect Wallet

Address: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A

Role: None

Profile

Address: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A

Role: None

Profile Info: Please register as either a Patient or Doctor to begin using the system.

Register As:

Patient Doctor

View Medical Records

Fetch Records

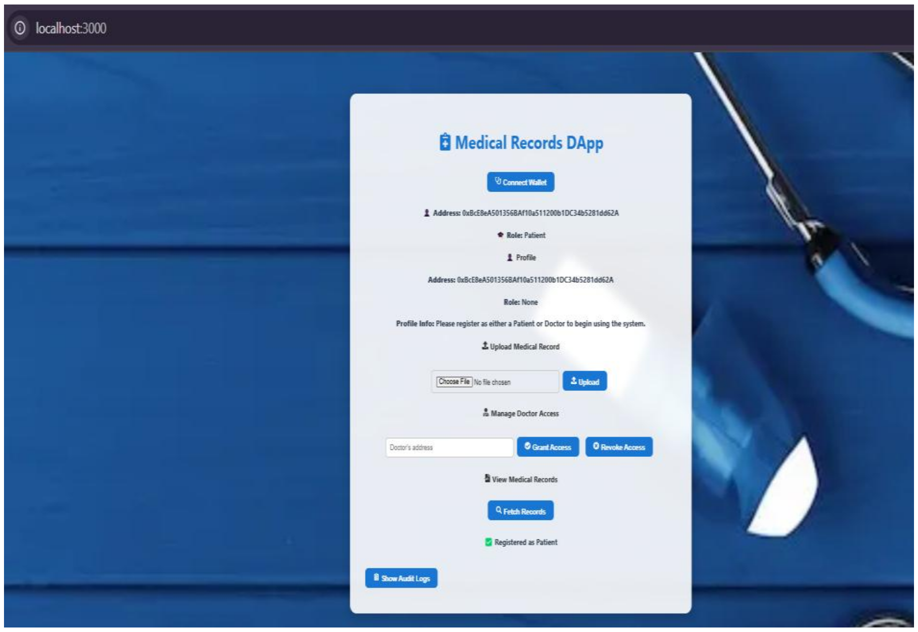
Connected: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A (None)

Show Audit Logs

Initially,

Patient window:

Running at port:3000



localhost:3000

Medical Records DApp

Connect Wallet

Address: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A

Role: Patient

Profile

Address: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A

Role: None

Profile Info: Please register as either a Patient or Doctor to begin using the system.

Upload Medical Record

Choose File No file chosen Upload

Manage Doctor Access

Doctor's address Grant Access Revoke Access

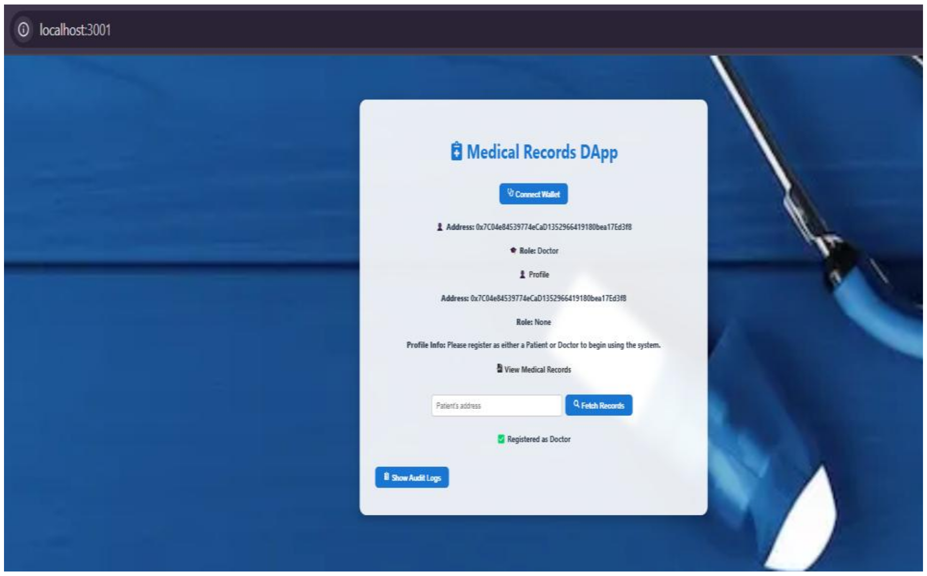
View Medical Records

Fetch Records

Registered as Patient

Show Audit Logs

Doctor window:



Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: Doctor

Profile

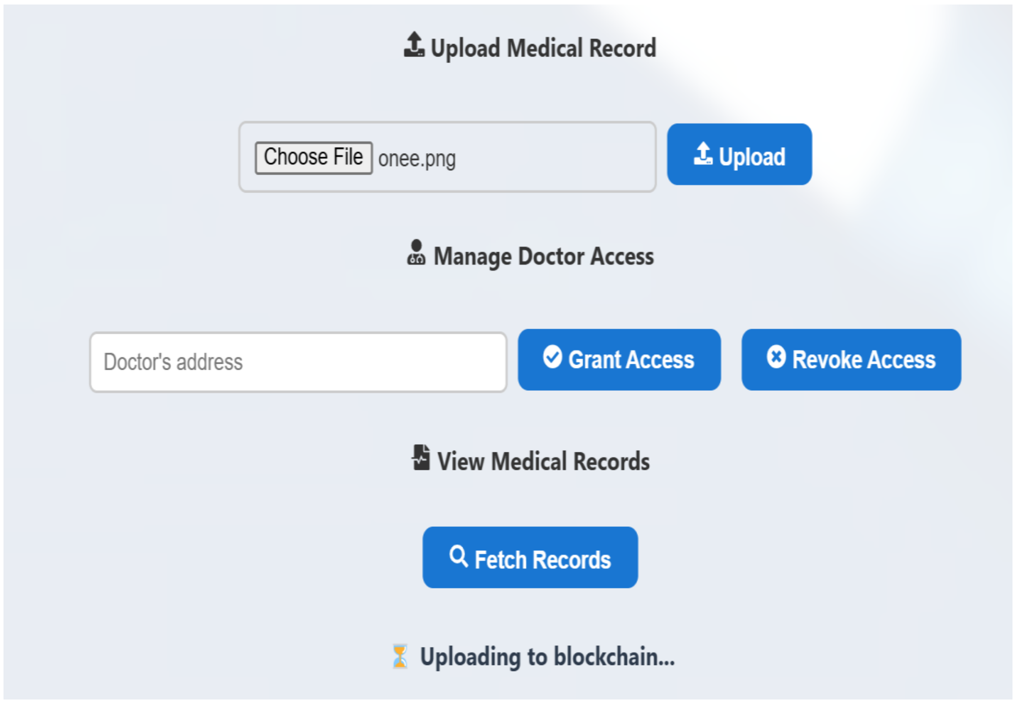
Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: None

Profile Info: Please register as either a Patient or Doctor to begin using the system.

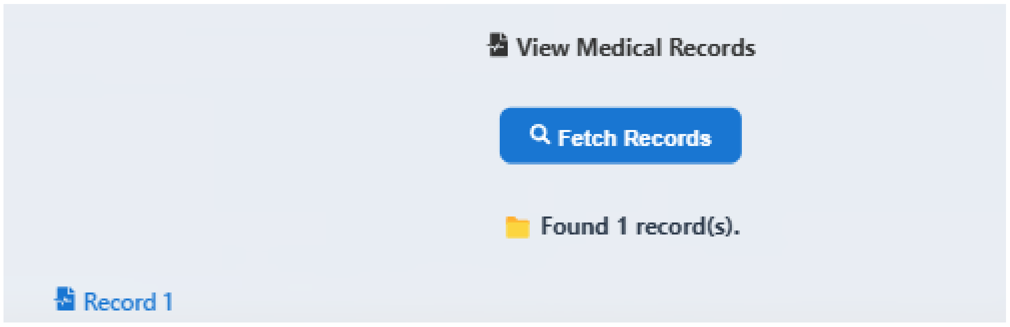
Registered as Doctor

Upload a file using the UI.

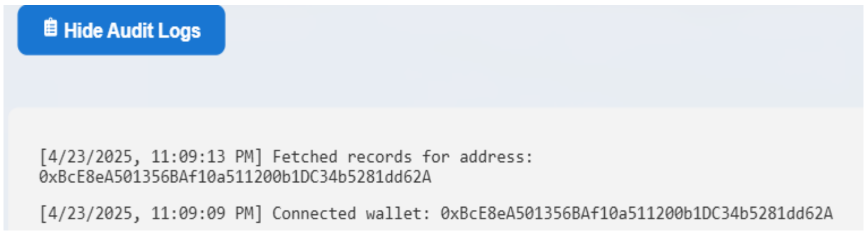


Uploading to blockchain...

Fetching the uploaded record:



Check console → see IPFS URL.

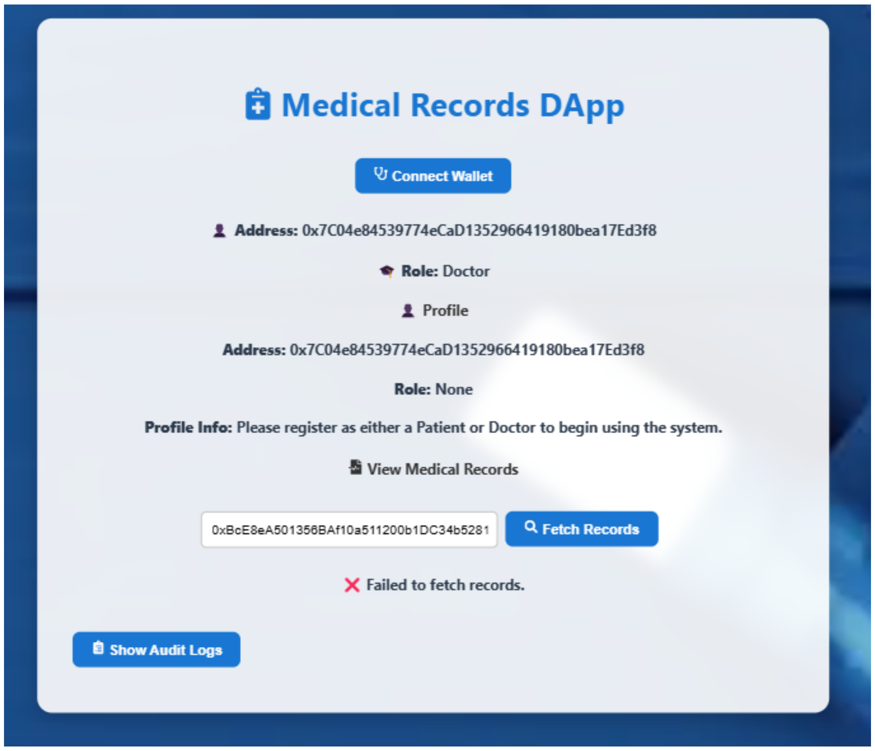


```
[4/23/2025, 11:09:13 PM] Fetched records for address:
0xBcE8eA501356BAf10a511200b1DC34b5281dd62A
[4/23/2025, 11:09:09 PM] Connected wallet: 0xBcE8eA501356BAf10a511200b1DC34b5281dd62A
```

Use MetaMask to confirm the transaction.
Hash stored in smart contract, retrievable via getFiles()

For the Doctor to view a patient's record, he/she needs to have the permission granted by the patient.

Without the access trying to open a medical record:



Medical Records DApp

Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: Doctor

Profile

Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: None

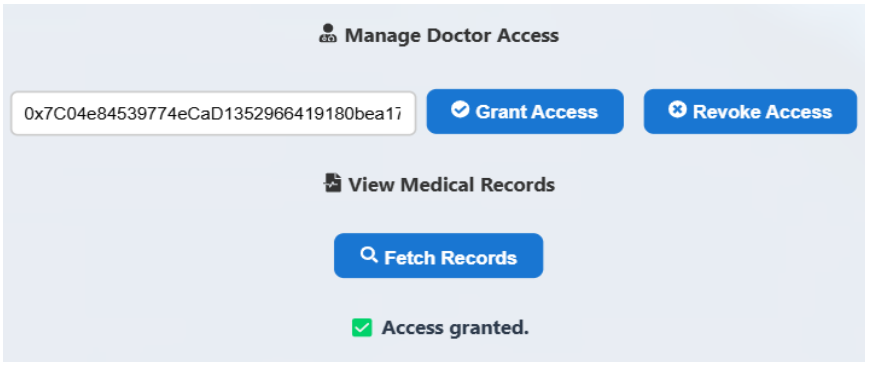
Profile Info: Please register as either a Patient or Doctor to begin using the system.

View Medical Records

0xBcE8eA501356BAf10a511200b1DC34b5281

Failed to fetch records.

Patient grants access to a particular doctor to view his records:



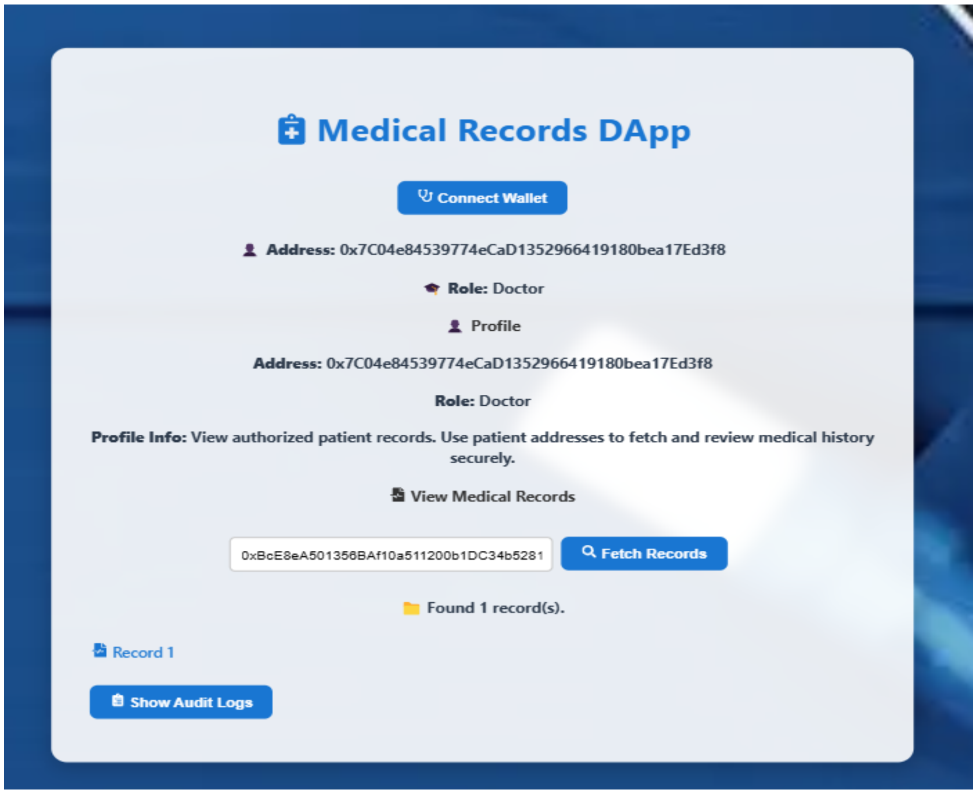
Manage Doctor Access

0x7C04e84539774eCaD1352966419180bea17

View Medical Records

Access granted.

Later trying with doctor to view patient's record:



Medical Records DApp

Connect Wallet

Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: Doctor

Profile

Address: 0x7C04e84539774eCaD1352966419180bea17Ed3f8

Role: Doctor

Profile Info: View authorized patient records. Use patient addresses to fetch and review medical history securely.

View Medical Records

0xBcE8eA501356BAf10a511200b1DC34b5281 Fetch Records

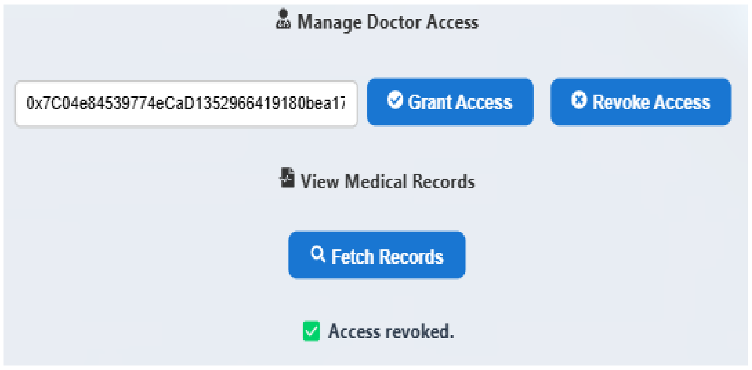
Found 1 record(s).

Record 1

Show Audit Logs

We can now see that the record is being displayed under the Doctor's dashboard.

We can also revoke the access whenever necessary:



Manage Doctor Access

0x7C04e84539774eCaD1352966419180bea17 Grant Access Revoke Access

View Medical Records

Fetch Records

Access revoked.

Now again, the doctor won't be able to access the data, which in turn helps the patient to maintain his privacy and share the required documents whenever necessary.

7. SECURITY AND PRIVACY CONSIDERATIONS

- **Data Immutability**: Once written, medical records cannot be altered.
- **Permissioned Access**: Only authorized users can view patient records, enforced via smart contract access modifiers.
- **Transparency**: Actions are publicly visible on the blockchain.
- **Future Enhancements**:
  - Off-chain encryption for sensitive data storage
  - Use of zero-knowledge proofs (zk-SNARKs) to validate data access rights without exposing contents

8. FUTURE ENHANCEMENTS

- **Timed Access Control:** Offering temporary access to healthcare providers, like allowing a doctor to access a patient's records for just 24 hours, ensuring data is available when needed but only for a limited time.
- **Specialization Filtering**: Allowing patients to search for doctors based on their specific medical specialty, making it easier to find the right expert for their needs.
- **Emergency Access Codes**: Giving emergency responders quick, read-only access to patient records during critical situations, ensuring they have important health information in real-time while protecting privacy.
- **Record Archival**: Creating a way to archive or hide older records, keeping the dashboard clean and making it easier for patients and doctors to focus on current medical information.
- **Audit Logs Interface**: Offering an easy-to-use interface that shows a history of who accessed the records and what changes were made, helping build trust and transparency in the system.

9. CONCLUSION

The Personalized Medical Vault app is a proof-of-concept that demonstrates how blockchain can be applied to securely manage healthcare data. With smart contracts handling the logic for data ownership and access permissions, this system mitigates many issues faced by centralized databases such as single-point failures and unauthorized tampering.

In the future, this Vault could integrate with national health systems, allowing cross-hospital data sharing with complete patient consent and traceability. Further work includes encrypted off-chain storage, advanced identity verification, and real-world deployment integration with hospital IT infrastructures.