Capstone Group IV- Project

Medical Record Systems DAPP

Contributors

Jean-Pierre Koudifo
Angela Maturo
Sackor Saydee
Alex Eakins

PROJECT OVERVIEW

OVERVIEW

The medical field is one of the largest industries in the world. However, innovation is very limited due to regulations. We decided to target this field due to potential benefits between patients and doctors sharing medical records and protecting patients privacy. We will show how Blockchain can revolutionize the Medical Records Industry.

We have created a system that will store patient medical records using smart contracts utilizing the Blockchain and connect the front end with Streamlit. We built a decentralized App called the “ Medical Record System” that will allow patients to tokenize their medical records on the blockchain. This DAPP will help eliminate HIPPA concerns due to the sensitivity of the data.

Brief Why!

The Travelers story:

Life Crisis/Incident

Billions of dollars spent in the US on protecting medical record.

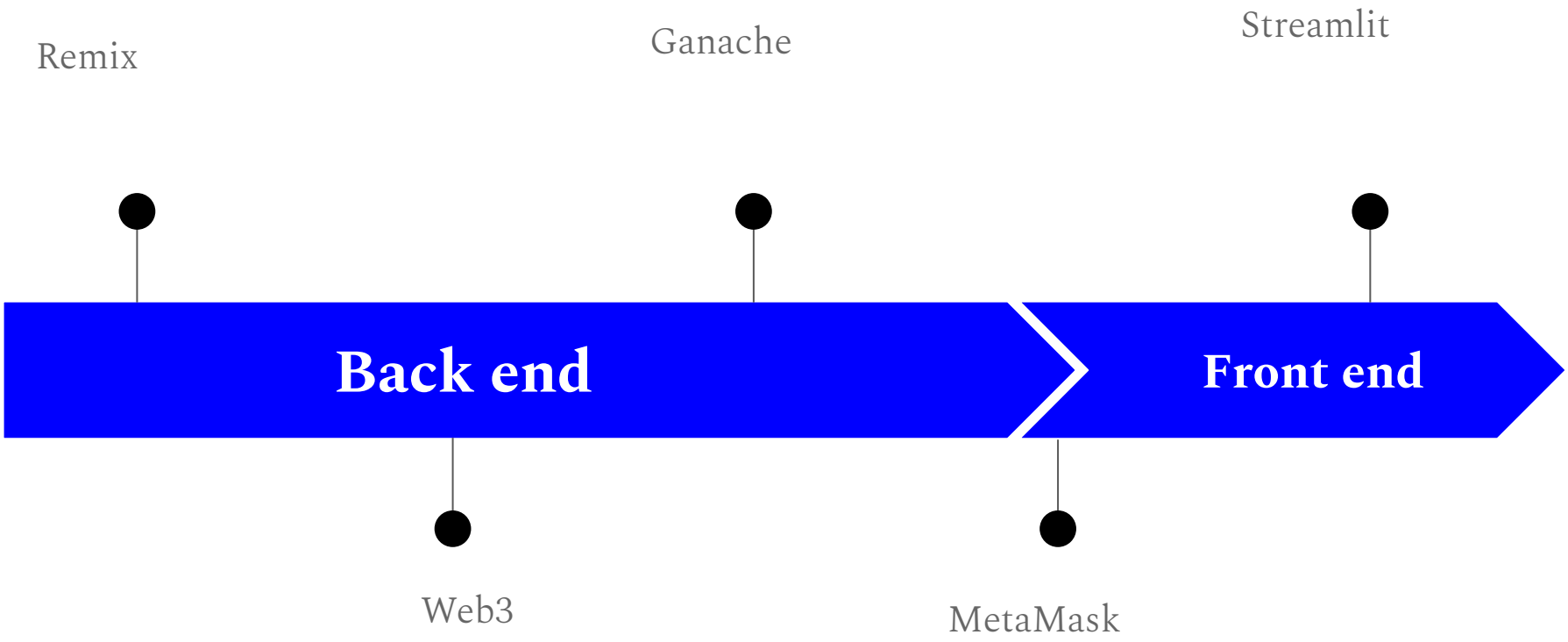
Major Issues in the healthcare/medical record industry

^ upward trend over the past 10 years

recent easing or reduction due to better policies and the use of encryption have reduced easily preventable breaches

According to hippaajournal.com

Technical Requirements


Remix
Ganache
Streamlit
Back end
Front end
Web3
MetaMask


Smart Contract

Process

Step 1-Smart Contract:

The back end

MedicalRecordToken

We created and compiled **MedicalRecordToken** using ERC20, ERC20Detailed, ERC20Mintable

MedicalRecord Smart Contract

We Created, compiled and deployed a **MedicalRecord** Smart Contract with ERC721Full

Loaded the contract

We Loaded the **MedicalRecord's** compiled ABI JSON file

Demo
Behind the Code

MedicalRecordToken Successfully deployed

SOLIDITY COMPILER

COMPILER

0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile MedicalRecordToken.sol

CONTRACT

MedicalRecordToken (MedicalRecordT

Publish on Ipfs

Publish on Swarm

Compilation Details

ABI Bytecode

Home | Medical_record.sol | MedicalRecordToken.sol

```
pragma solidity ^0.5.0;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC20/ERC20.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC20/ERC20Detailed.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC20/ERC20Mintable.sol";
contract MedicalRecordToken is ERC20, ERC20Detailed, ERC20Mintable {
    constructor(
        string memory name,
        string memory symbol,
        uint initial_supply
    )
        ERC20Detailed(name, symbol, 18)
        public
    {
        // constructor can stay empty
    }
}
```

COMPILER

0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE

Solidity

EVM VERSION

default

COMPILER CONFIGURATION

Auto compile

Enable optimization 200

Hide warnings

Compile Medical_record.sol

CONTRACT

MedicalRecord (Medical_record.sol)

Publish on Ipfs

Publish on Swarm

Compilation Details

ABI Bytecode

```
pragma solidity ^0.5.0;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC721/ERC721Full.sol";
contract MedicalRecord is ERC721Full {
    constructor() public ERC721Full("MedicalRecordToken", "MRT") {}
    struct PatientRecord {
        string HospitalName;
        string DoctorName;
        uint Age;
        uint Height;
        uint Weight;
        string MedicalHistory;
        string MedicalExam;
        uint256 RecordDate;
    }
    mapping(uint256 => PatientRecord) public MedicalDataBank;
    event SavePatientRecord(uint256 tokenId, string HospitalName,
        string DoctorName,
        uint Age,
        uint Height,
        uint Weight,
        string MedicalHistory,
        string MedicalExam,  uint256 RecordDate);

    function setPatientRecord(
        address Patient,
        string memory HospitalName,
        string memory DoctorName,
        uint Age,
        uint Height,
        uint Weight,
        string memory MedicalHistory,
        string memory MedicalExam,
        uint256 initialRecordDate,
        string memory tokenURI
    ) public returns (uint256) {
        uint256 tokenId = totalSupply();
        _mint(Patient, tokenId);
        _setTokenURI(tokenId, tokenURI);
        MedicalDataBank[tokenId] = PatientRecord(HospitalName, DoctorName, Age, Height, Weight, MedicalHistory, MedicalExam, initialRecordDate);
        emit SavePatientRecord(tokenId, HospitalName, DoctorName, Age, Height, Weight, MedicalHistory, MedicalExam, initialRecordDate);
        return tokenId;
    }
}
```

Patient Record

0 listen on network Search with transaction hash or address

creation of MedicalRecord pending...

creation of MedicalRecord pending...

[block:2 txIndex:0] from: 0x299...B32fE to: MedicalRecord.(constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0x457...46061

MedicalRecord Smart Contract Compiled Successfully

Medical Record - Smart Contract Deployed

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Injected Web3

Custom (5777) network

ACCOUNT: 0x299...B32fE 99.8744524E

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: MedicalRecord - contracts/Project Cap

Deploy

Publish to IPFS

OR

At Address | Load contract from Address

Transactions recorded 3

Deployed Contracts

MEDICALRECORD AT 0XBDE...0E975 (B

Injected Web3

Home | Medical_record.sol | MedicalRecordToken.sol

```
pragma solidity ^0.5.0;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC721/ERC721Full.sol";
contract MedicalRecord is ERC721Full {
    constructor() public ERC721Full("MedicalRecordToken", "MRT") {}
    struct PatientRecord {
        string HospitalName;
        string DoctorName;
        uint Age;
        uint Height;
        uint Weight;
        string MedicalHistory;
        string MedicalExam;
        uint256 RecordDate;
    }
    mapping(uint256 => PatientRecord) public MedicalDataBank;
    event SavePatientRecord(uint256 tokenId, string HospitalName,
        string DoctorName,
        uint Age,
        uint Height,
        uint Weight,
        string MedicalHistory,
        string MedicalExam,  uint256 RecordDate);

    function setPatientRecord(
        address Patient,
        string memory HospitalName,
        string memory DoctorName,
        uint Age,
        uint Height,
        uint Weight,
        string memory MedicalHistory,
        string memory MedicalExam,
        uint256 initialRecordDate,
        string memory tokenURI
    ) public returns (uint256) {
        uint256 tokenId = totalSupply();
        _mint(Patient, tokenId);
        _setTokenURI(tokenId, tokenURI);
        MedicalDataBank[tokenId] = PatientRecord(HospitalName, DoctorName, Age, Height, Weight, MedicalHistory, MedicalExam, initialRecordDate);
        emit SavePatientRecord(tokenId, HospitalName, DoctorName, Age, Height, Weight, MedicalHistory, MedicalExam, initialRecordDate);
        return tokenId;
    }
}
```

0 listen on network | Search with transaction hash or address

creation of MedicalRecord pending...

[block:2 txIndex:0] from: 0x299...B32fE to: MedicalRecord.(constructor) value: 0 wei data: 0x608...10032 logs: 0 hash: 0x457...46061

creation of MedicalRecord pending...

Debug

MetaMask

Account 9 → New Contract

https://remix.ethereum.org

CONTRACT INTERACTION

DETAILS | DATA

EDIT

Estimated gas fee 0.06277376

0.062774 ETH

Site suggested Max fee: 0.06277376 ETH

Total 0.06277376

0.06277376 ETH

Amount + gas fee Max amount: 0.06277376 ETH

Reject | Confirm

We use the Remix IDE, MetaMask, and Ganache.

Medical Record - Successfully Metadata published

← BACK

TX 0×e338d020365e0463a891336b5c901c2323f1ddaeb89747b76cb2c684b44bff6a

SENDER ADDRESS

0×29978fb187b32898a91F0df7a2FB43fB106B32fE

CREATED CONTRACT ADDRESS

0×679851126BE007e31A19f596802B351c1967d3b4

CONTRACT CREATION

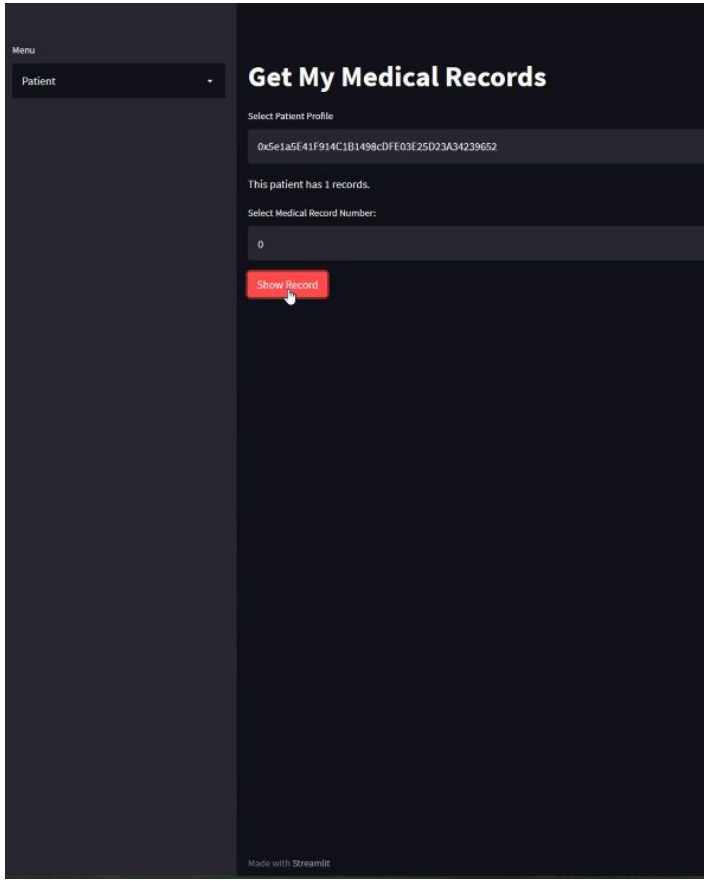
VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	3138688	20000000000	3138688	1

Contract Address

TX DATA

0×6080604052348015620000115760008 0fd5b5060405180604001604052806012815260020017f4d65646963616c5265636f7264546f6b656e0000000000000000000000
0000081525060405180604001604052806003815260020017f4d52540000000000000000000000000000000000000000000000000000000000000000081525081816200009863 01ff
c9a760e01b6200011c60201b60201c565b620000b06380ac58cd60e01b6200011c60201b60201c565b620000c863780e9d6360e01b6200011c60201b60201c565b8160099
0805190602001906200 00e0929190620002255565b5080600a90805190602001906200 00f9929190620002255565b5062000112635b5e139f60e01b6200011c60201b60201c
565b50505050620002d4565b63ffffffff60e01b817bffffffffffffffffffffffffffffffffffffffffffffffffffffffff1916141562000 1b9576040517f08c379a0000
0000000000000000000000000000000000000000000000000000000081526004018080602001828103825260 1c8152602001807f4552433136353a20696e76616c696420696e
746572666163652069640000000081525060200191505060405180910390fd5b60001600080837bffffffffffffffffffffffffffffffffffffffffffffffffffffffff191
67bffffffffffffffffffffffffffffffffffffffffffffffffffffffff191681526020019081526020016000206000610 1000a81548160ff021916908315150217905550
50565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f1062000268578051 60ff191683800117855562000299 5
65b828001600101855582156200029957918201 5b828111156200029857825182559160200191906001019062000 27b565b5b509050620002a89190620002ac565b509056
5b620002d191905b808211156200 02cd576000816000905550600101620002b3565b5090565b90565b61357a80620002e46000396000f3fe6080604052348015610010576
00080fd5b506004361061012c5760003560e01c80636352211e116100ad57806395d89b4111610071578063 95d89b4114610cea578063a22cb46514610d6d578063b88d4f
de14610dbd578063c87b56dd14610ec2578063e985e9c514610f695761012c565b80636352211e1461055657806368 75daf1146105c45780636c0360eb146108225780637
0a08231146108a5578063897143 6b146108fd5761012c565b806323b872dd116100f457806323b872dd146102f35780632f745c591461036157806342842e0e146103c357
80634f6ccce7146104315780635 7a622e2146104735761012c565b806301ffc9a7146101315780630 6fdde0314610196578063081812fc1461021957806309 5ea7b314610
28757806318160ddd146102d5575b600080fd5b61017c6004803603602081101561014757600080fd5b810190808035 7bffffffffffffffffffffffffffffffffffffffff
ffffffffffffffff191690602001909291905050610fe5565b604051808215151515815260200191505060405180910390f35b61019e61104c565b604051808060200 18
2810382528381815181526020019150805190602001908083836005b838110156101de57808201518184015260208101905061 01c3565b50505050905090810190601f16
801561020b578082038051600183602003610100 0a031916815260200191505b509250505060405180910390f35b6102456004803603602081101561022f57600080fd5b8
101908080359060200190929190505050611 0ee565b604051808273ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff
ff16815260200191505060405180910390f35b6102d36004803603604081101561029d57600080fd5b810190808035 73ffffffffffffffffffffffffffffffffffffffff1
69060200190929190803590602001909291905050506111895 65b005b6102dd611370565b604051808281526020019150506040518091 0390f35b61035f60048036036060
81101561030957600080fd5b810190808035 73ffffffffffffffffffffffffffffffffffffffff169060200190929190803573ffffffffffffffffffffffffffffffffffffffff
ffffff169060200190929190803590602001909291905050506113 7d565b005b6103ad60048036036040811015610377576000 80fd5b810190808035 73ffffffffffffffff
ffffffffffffffffffffffff169060200190929190803590602001909291905050506113 3565b604051808281526020019150506040518091 0390f35b61042f600480360

Contract deployed in Garnache



Process

Step 2 - Streamlit app: The Front End

Front-End Development

Streamlit

Libraries & Contracts

```python
from datetime import date
import json
import os
from pathlib import Path
from datetime import datetime

import apps
import streamlit as st
from dotenv import load_dotenv
from web3 import Web3
from pinata import *

w3 = Web3(Web3.HTTPProvider(os.getenv("WEB3_PROVIDER_URI")))


##################################################################################
# Load Contract
##################################################################################

load_dotenv("web3.env")

def load_contract():

    # Load ABI
    with open(Path('./contracts/compiled/medrecord.json')) as f:
        contract_abi = json.load(f)

    contract_address = os.getenv("SMART_CONTRACT_ADDRESS")

    contract = w3.eth.contract(
        address=contract_address,
        abi=contract_abi
    )

    return contract


contract = load_contract()

```

Creating Pin Charts & Example Profiles

```python
##########################################################################################
# Helper functions to pin files and json to Pinata
##########################################################################################


def pin_chart(pat_address, hospital_input, docname_input, age_input, height_input, weight_input, med_his_notes,
exam_notes, record_date):

    # Build a token metadata file for the artwork
    pa_file_hash = pin_file_to_ipfs(pat_address)
    hi_file_hash = pin_file_to_ipfs(hospital_input)
    di_file_hash = pin_file_to_ipfs(docname_input)
    ai_file_hash = pin_file_to_ipfs(str(age_input))
    hi_file_hash = pin_file_to_ipfs(str(height_input))
    wi_file_hash = pin_file_to_ipfs(str(weight_input))
    mh_file_hash = pin_file_to_ipfs(med_his_notes)
    en_file_hash = pin_file_to_ipfs(exam_notes)
    rd_file_hash = pin_file_to_ipfs(record_date)

    token_json = {
        "patient": pa_file_hash,
        "hospital": hi_file_hash,
        "doctor": di_file_hash,
        "age": ai_file_hash,
        "height": hi_file_hash,
        "weight": wi_file_hash,
        "medical history": mh_file_hash,
        "exam notes": en_file_hash,
        "record date": rd_file_hash,
    }
    json_data = convert_data_to_json(token_json)

    # Pin the json to IPFS with Pinata
    json_ipfs_hash = pin_json_to_ipfs(json_data)

    return json_ipfs_hash
```

```python
##########################################################################################
# Established Patient/Doctor Profiles for Example
##########################################################################################

patient_1_address = "0x10dfC6C4b40Ff39882E8A107E432305E39dE55d4"
patient_2_address = "0x5e1a5E41F914C1B1498cDFE03E25D23A34239652"
patient_test_address = "0xc3804461E5BE1D91c8Dc41E0cb26CE30d6654A95"

doctor_1_address = "0xbE8DE506e9b48627D5B47703b52E0E8249802a71"
doctor_2_address = "0x1ceA88Ab386170eB43Ecbd4e8b983C5433cAAb25"

d_accounts = [doctor_1_address, doctor_2_address]
p_accounts = [patient_1_address, patient_2_address, patient_test_address]
```

Main Menu

```
def to_integer(dt_time):
    return 10000*dt_time.year + 100*dt_time.month + dt_time.day

def main():
    menu = ['Home', 'Doctor', 'Patient']
    choice = st.sidebar.selectbox("Menu", menu)
    if choice == 'Home':
        st.title("Welcome to BlocDoc")
        st.write("BlocDoc is an Electronic Health Record Decentralized Application (EHR-dApp), an etherium-based web application that helps
        patients and doctors manage electronic health records in a decentralized format. We are trying to apply the EHR systems logic to Blockchain
        architecture, of course, on a proof of concept level.")
        st.markdown("---")
```

Doctor

```python
elif choice == 'Doctor':
    st.title("Doctor EMR")
    emr_form = st.container()
    with emr_form:
        with st.form('form1'):
            pat_address = st.selectbox("Select Patient", options=p_accounts)
            hospital_input = st.text_input(label='Hospital Name')
            docname_input = st.text_input(label='Doctor Name')
            age_input = st.number_input(label='Age', step=1)
            height_input = st.number_input(label='Height in Inches', step=1)
            weight_input = st.number_input(label='Weight', step=1)
            med_his_notes = st.text_input(
                label='Notes of Patient Medical History')
            exam_notes = st.text_input(label='Examination Notes')
            record_date = to_integer(datetime.now())
            dsubmit_button = st.form_submit_button(label='Submit')

            if dsubmit_button:
                chart_ipfs_hash = pin_chart(pat_address,hospital_input, docname_input, age_input, height_input, weight_input, med_his_notes,
                exam_notes, record_date)

                chart_uri = f"ipfs://{chart_ipfs_hash}"
                tx_tokenID = contract.functions.setPatientRecord(
                    pat_address,
                    hospital_input,
                    docname_input,
                    age_input,
                    height_input,
                    weight_input,
                    med_his_notes,
                    exam_notes,
                    record_date,
                    chart_uri
                )
               #  st.write("TOKENID:", tx_tokenID)
                tx_hash = tx_tokenID.transact({'from': doctor_1_address, 'gas': 1000000})
               #  st.write("HASH:", tx_hash)
                receipt = w3.eth.waitForTransactionReceipt(tx_hash)
                st.write("Transaction Receipt Mined:")
                st.write(dict(receipt))
                st.write(
                    "You can view the pinned metadata file with the following IPFS Gateway Link")
                st.markdown(
                    f"[Chart IPFS Gateway Link](https://ipfs.io/ipfs/{chart_ipfs_hash})")
        st.markdown("---")
```

Patient

```
    elif choice == 'Patient':
        st.title("Get My Medical Records")
        my_patient_address = st.selectbox(label='Select Patient Profile', options=p_accounts)
        tokens = contract.functions.balanceOf(my_patient_address).call()
        st.write(f"This patient has {tokens} records.")


        token_id = st.selectbox("Select Medical Record Number:", list(range(tokens)))

        if st.button("Show Record"):
            token_uri = contract.functions.tokenURI(token_id).call()
            history_filter = contract.events.SavePatientRecord.createFilter(
                fromBlock=0, argument_filters={"tokenId": token_id}
)

            reports = history_filter.get_all_entries()

            if reports:
                for report in reports:
                    report_dictionary = dict(report)
                    st.markdown("### Medical History Report Log")
                    OutputData = report_dictionary["args"]
                    st.write(f"Doctor Name: {OutputData.DoctorName}")
                    st.write(f"Hospital Name: {OutputData.HospitalName}")
                    st.write(f"Age: {OutputData.Age}")
                    st.write(f"Height: {OutputData.Height} inch")
                    st.write(f"Weight: {OutputData.Weight} lbs")
                    st.write(f"Medical History: {OutputData.MedicalHistory}")
                    st.write(f"Examination Notes: {OutputData.MedicalExam}")
                    for i in range(3):
                        st.write("")
                    st.write(f'*last updated at {datetime.strptime(str(OutputData.RecordDate),"%Y%m%d")}*')
                    #st.write(stringData.DoctorName)

                    #st.write(outputData)



# Functions
main()
```

Video Demo

Summary:

How the final project should work:

Challenges we had: connecting the back and front of the contract.

Implementing the login

Implementing Hydralit library(our original frontend library

Connecting the front and backend

QUESTIONS?

We have answers...

Resources/References

hipaajournal.com/healthcare-data-breach-statistics/

https://www2.deloitte.com/us/en/pages/public-sector/articles/blockchain-opportunities-for-health-care.html