PROJECT DESIGN PHASE & SOLUTION ACHITECTURE

Date	30 November 2023
Team ID	NM2023TMID04547
Project Name	Blockchain Technology For Electronic Health Records
Maximum Marks	4 Marks

Solution Architecture:
Before the introduction of smart contacts on the blockchain, the main discussions on Electronic Health Record (EHR) Management focused on whether to use cloud infrastructures or local centralized systems for storing and sharing EHRs. These centralized systems implied that each hospital and healthcare company would have to keep data on premises in locally managed structures and databases.

A blockchain is a data structure where the records are stored in a linked sequence of blocks. This sequence forms a distributed ledger, which means it is replicated in multiple machines, called nodes, that communicate with one another. The nodes form a peer-to-peer network where every update to the ledger must be accepted by the network using a consensus protocol. The consensus protocol assures that everybody has the same view on the status of the system.

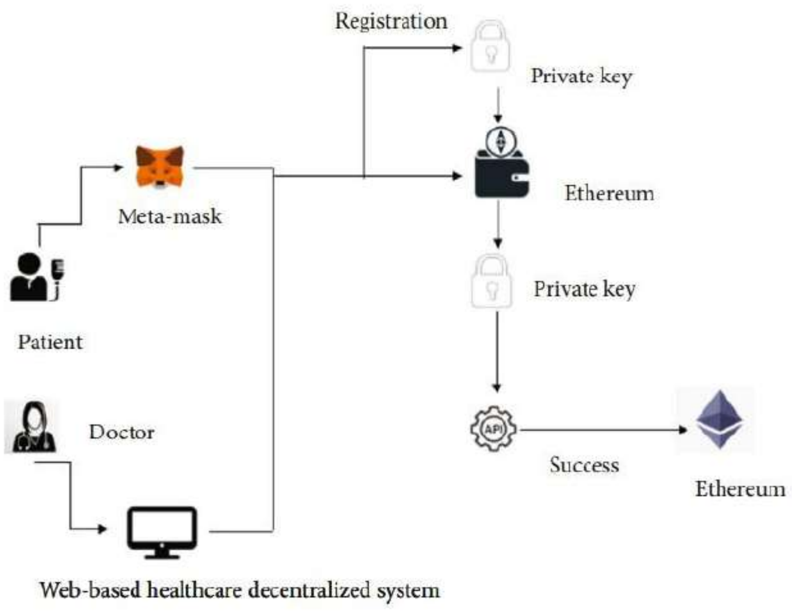
After the design and implementation of a basic EHRs management system and the execution of a set of test cases, it will be possible to discuss the benefits and trade-offs that the system entails. The discussion will focus on the performance of a permissioned blockchain for EHRs management. Normal and disaster scenarios will be compared using the following indicators to get important insights on how a crisis affects the operations of a blockchain network:

• Success rate: the number of successes and failures of a batch of requests. It is important to limit the number of failures caused by a surge of requests during an emergency;

• Transaction commit and read latency: this refers to the time it takes for the blockchain-based system to process an access request to an EHR in a disaster situation. It is important as timeliness in getting health data, especially in emergencies, is critical.

• Transaction commit and state read throughput (TPS): this refers to the number of requests that can be managed by the system at the same time. Being able to access and modify a growing number or request is essential to enable everybody to interact with the system;

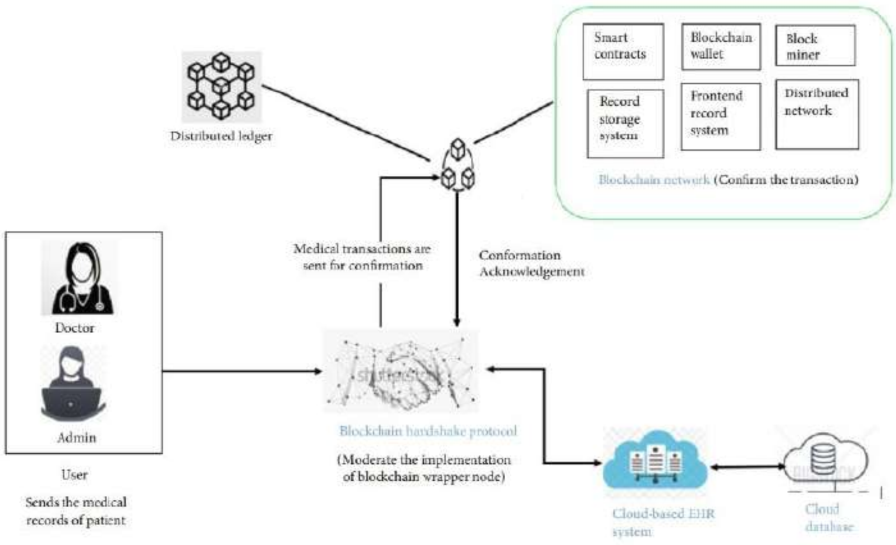
• Resource consumption (CPU, Memory and network IO): it is necessary to take these parameters into account as they affect all the other indicators.



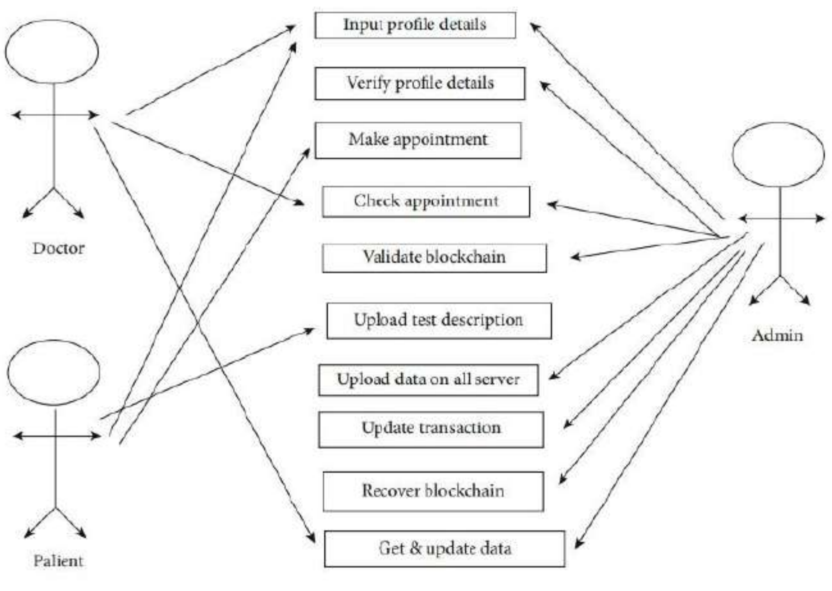
6.2 Sprint Planning & Estimation:

Block Diagram illustrates the block diagram. Our proposed design has four major components: a user application, a blockchain handshake protocol, a cloud, and a public blockchain network. The system is a virtual representation that serves two purposes. For starters, it provides users with access to application interfaces. Doctors and system administrators are two types of users in our system. Each user has a distinct function. As a result, the user application delivers different user interfaces depending on the user role. Second, based on the data entered by the user, the user application creates an initial transaction. For the purpose of confirmation, the transaction is submitted to the blockchain handshake protocol. Finally, a user interface establishes the relationship between users and the blockchain handshake protocol. The proposed architecture's fundamental component is the blockchain handshake (BH) protocol. This component connects the database server, the blockchain network, and the cloud-based health record system, which acts as a wrapper. This proposed architecture makes use of the Ethereum blockchain network. A distributed ledger that connects blockchain nodes is known as the public blockchain network. Blockchain nodes are miners who are in charge of updating the blockchain based on the decision method. Alternatively, blockchain nodes accept transactions and use the network's smart contracts to authenticate them.

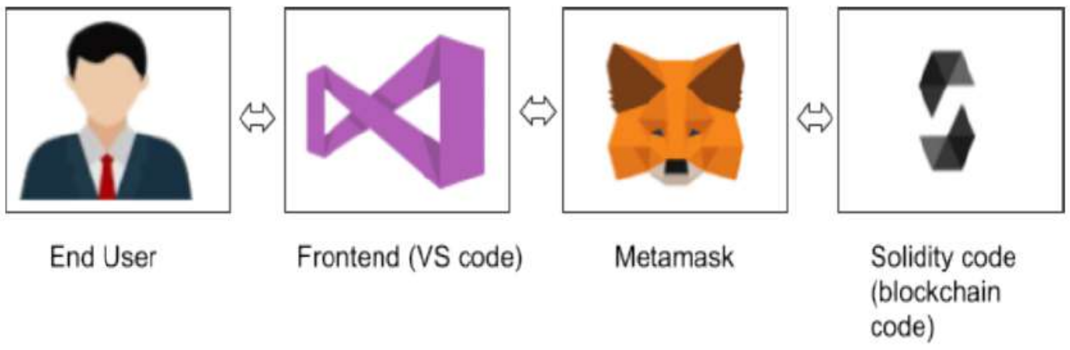
Block diagram of the blockchain-based EHR system

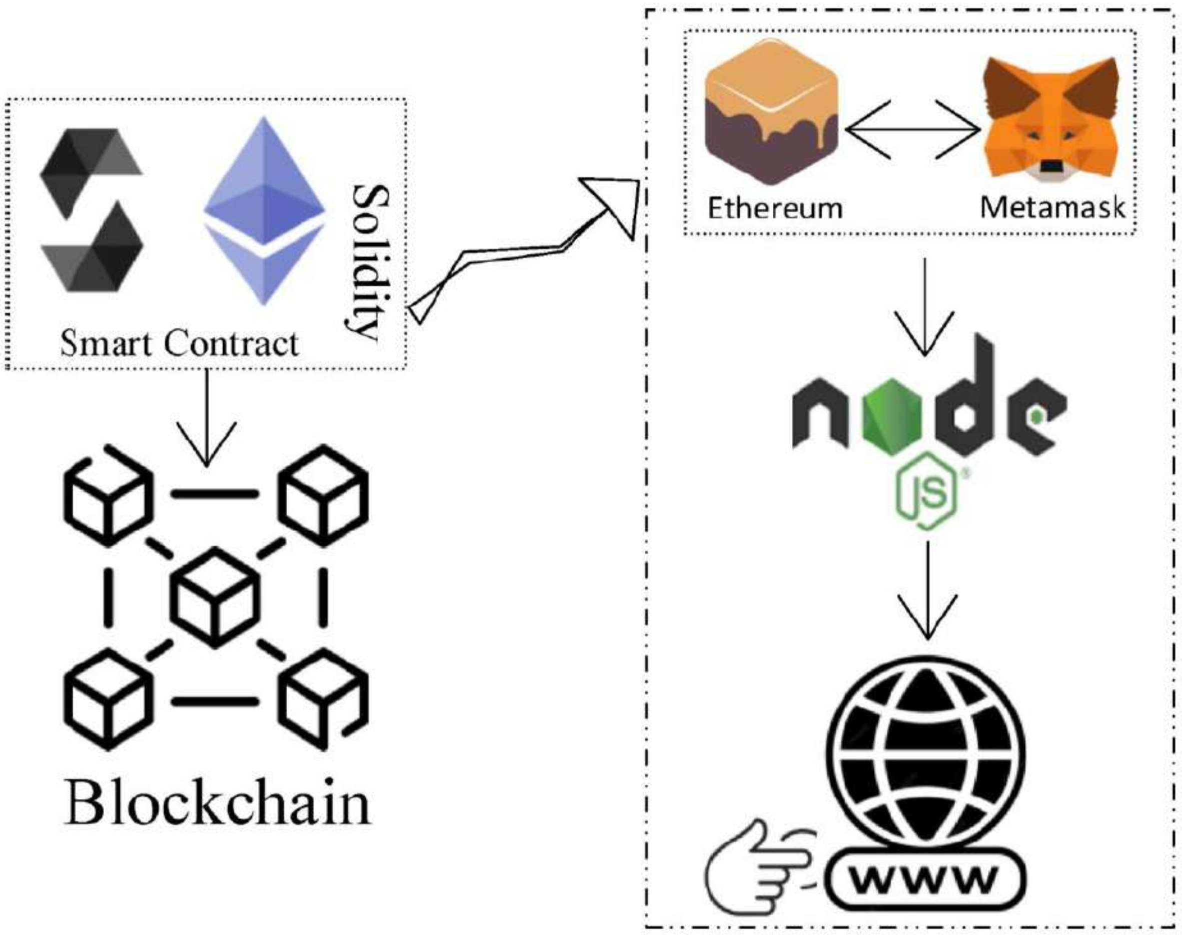

Distributed ledger
Smart contracts
Blockchain wallet
Block miner
Record storage system
Frontend record system
Distributed network
Blockchain network (Confirm the transaction)
Medical transactions are sent for confirmation
Conformation Acknowledgement
Doctor
Admin
User
Sends the medical records of patient
Blockchain handshake protocol
(Moderate the implementation of blockchain wrapper node)
Cloud-based EHR system
Cloud database

Use case diagram of the EHR system



Solution Architecture Diagram

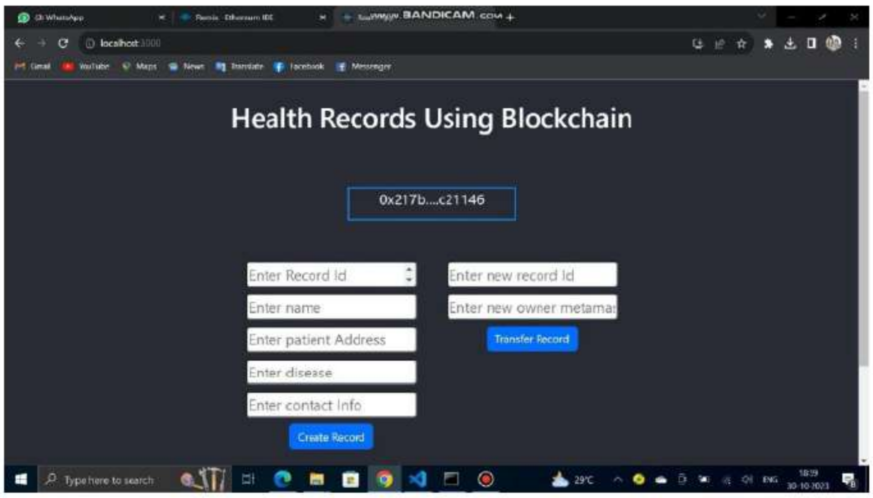




Output Screenshots

Local Host IP address : http://localhost:3000/

Screenshot of output



Video Demo Link : **https://youtu.be/cNsyMfDFRks?feature=shared**

Source code 1 :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract HealthRecords
{
        struct PatientRecord
  {
        String Name;
         address patientAddress;
        string dieses;
        string contactInfo;
   }
        mapping(uint256 => PatientRecord) public records;
        event RecordCreated(uint256 indexed recordId, address indexed patientAddress);
        event RecordTransferred
(
        uint256 indexed recordId,
            address indexed from,
            address indexed to
   );
 modifier onlyOwner(uint256 recordId)
{
     require(msg.sender == records[recordId].patientAddress,"Only contract owner can call this");
       _;
  }
function createRecord
(
           uint256 recordId,
           string memory name, address _patientAddress, string memory _diseases, string memory
           _contactInfo
   )
 external {
            records[recordId].Name = name;
            records[recordId].patientAddress = _patientAddress;
            records[recordId].dieses = _diseases;
           records[recordId].contactInfo = _contactInfo;
           emit RecordCreated(recordId, _patientAddress);
           }
           function transferRecord(uint256 recordId, address newOwner) external
onlyOwner(recordId)
           {
            //require(records[recordId].patientAddress == newOwner, "New Owner should have
different Address");
            require(records[recordId].patientAddress == msg.sender, "Only record owner can
transfer");
           records[recordId].patientAddress = newOwner;
           emit RecordTransferred(recordId, records[recordId].patientAddress, newOwner)
           }
           function getRecordData
           (
            uint256 recordId
```

```
        )
        external view returns (string memory, address, string memory,string memory) {
        return (records[recordId].Name,
        records[recordId].patientAddress,
        records[recordId].dieses,
         records[recordId].contactInfo);
         }
         function getRecordOwner(uint256 recordId) external view returns (address)
         {
      return records[recordId].patientAddress;
         }
}
```

Source code 2 :

```
const { ethers } = require("ethers");
const abi = [
 {
  "anonymous": false,
  "inputs": [
   {
    "indexed": true,
    "internalType": "uint256",
    "name": "recordId",
    "type": "uint256"
   },
   {
    "indexed": true,
    "internalType": "address",
    "name": "patientAddress",
    "type": "address"
   }
  ],
  "name": "RecordCreated",
  "type": "event"
```

```
},
{
 "anonymous": false,
 "inputs": [
  {
   "indexed": true,
   "internalType": "uint256",
   "name": "recordId",
   "type": "uint256"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "from",
   "type": "address"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "to",
   "type": "address"
  }
 ],
 "name": "RecordTransferred",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "recordId",
   "type": "uint256"
  },
  {
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
    "internalType": "address",
    "name": "_patientAddress",
    "type": "address"
   },
   {
    "internalType": "string",
    "name": "_diseases",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "_contactInfo",
    "type": "string"
   }
  ],
  "name": "createRecord",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
 },
 {
  "inputs": [
   {
    "internalType": "uint256",
    "name": "recordId",
    "type": "uint256"
   }
  ],
  "name": "getRecordData",
 "outputs": [
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "recordId",
   "type": "uint256"
  }
 ],
 "name": "getRecordOwner",
 "outputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "",
   "type": "uint256"
  }
 ],
 "name": "records",
 "outputs": [
  {
   "internalType": "string",
   "name": "Name",
   "type": "string"
  },
  {
   "internalType": "address",
   "name": "patientAddress",
   "type": "address"
  },
  {
   "internalType": "string",
   "name": "dieses",
   "type": "string"
   },
   {
    "internalType": "string",
    "name": "contactInfo",
    "type": "string"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [
   {
    "internalType": "uint256",
    "name": "recordId",
    "type": "uint256"
   },
   {
    "internalType": "address",
    "name": "newOwner",
    "type": "address"
   }
  ],
  "name": "transferRecord",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
 }
]

if (!window.ethereum) {
 alert('Meta Mask Not Found')
 window.open("https://metamask.io/download/")
}
```

```
export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0x8837a10cf07813729bCEB5381A6D435E986240d4"
export const contract = new ethers.Contract(address, abi, signer)
```