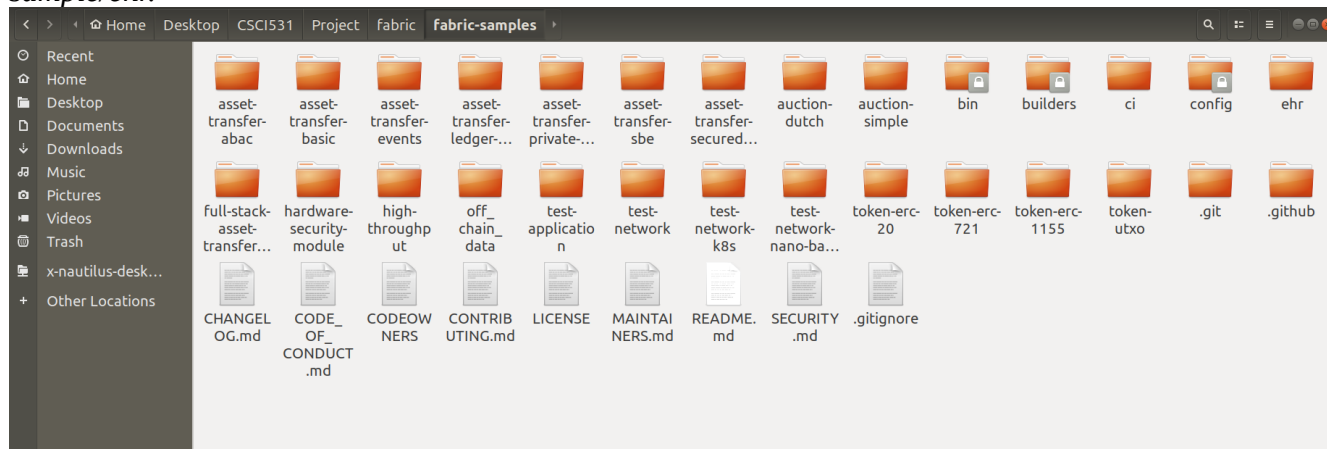


**Secure Decentralized EHR(Electronic health record) Audit System****Note:**

- This project implements **Option 2** using Hyperledger Fabric(blockchain technology) and a Node.js application for the frontend.
- The languages used are Javascript and HTML/CSS.
- The code for the Fabric smart contract/chaincode has been expanded using the stubs provided in *fabric-sample*(this repo is installed as the Hyperledger Fabric codebase and contains sample scripts).
- The code for the front end has been modified from a previous project I created for the web development course CSCI571. Apart from this, a Javascript code snippet for AES-256(CTR mode) cipher using JS library *crypto* has been taken from <https://stackoverflow.com/questions/60369148/how-do-i-replace-deprecated-crypto-createcipher-in-node-js>
- The code submission includes the folder named **ehr**. This folder **has to be placed inside the fabric-sample folder** after installing Hyperledger, and then the below instructions will help run the prototype.
- The project has been developed on Linux(Ubuntu 18.04).
- All external libraries used have been specified in the implementation section.

**Link to demo video:** [https://drive.google.com/file/d/1qMc3AAksdX9oSoGIzMx5gbZmE57g0r7J/view?usp=share\\_link](https://drive.google.com/file/d/1qMc3AAksdX9oSoGIzMx5gbZmE57g0r7J/view?usp=share_link)

View of the project top-level folder. I have installed Hyperledger(i.e. *fabric-samples* directory) in the folder *fabric*. This is the content of the Hyperledger workspace. The project resides in *fabric/fabric-sample/ehr*.



The program runs using 2 terminals. The first terminal is to be opened from *fabric/samples/test-network* (to setup the system), and the second from *fabric-samples/ehr/application-javascript*(to launch the web browser UI). The UI can be accessed from <http://localhost:3000>

**Software needed:**

Hyperledger Fabric <https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html>

Node.js (Instruction included in hyperledger install info)

Docker <https://docs.docker.com/engine/install/>

git

curl

## Run Instructions

To run the project(depending on your installation, may need to run as root or use *sudo* with every command), from inside the fabric folder(where hyperledger fabric has been downloaded):

1. Create network, organizations, orderer, the channel/ledger:

```
# cd fabric-samples/test-network
# ./network.sh down
# ./network.sh up createChannel -c mychannel -ca
```

2. Deploy the smart contract on the channel, peers:

```
# ./network.sh deployCC -ccn basic -ccp ../ehr/chaincode-javascript/ -ccl javascript
```

3. Start the Fabric application(the UI) from another terminal:

We will interact with the smart contract through a set of Node.js applications. Change into the *application-javascript* directory:

```
# cd fabric-samples/ehr/application-javascript
```

Install node.js and application specific dependencies. This will only be done once and will automatically install all external JS Libraries needed:

```
# npm install
# npm install -g nodemon
# npm install express --save
```

Every time we want to run the UI application( make sure the *wallet* folder if present is deleted from the *fabric-samples/ehr/application-javascript*. It will be there if application has been run before) do:

```
# node app.js
```

## System workflow

The distributed system is accessed using a front end UI that runs on a web browser through which users, either patients or auditors, access the EHR audit system where the audit records are stored on a public ledger shared by all users. The application connects to the audit system using a gateway on successful authorization of the user credentials. Please note that the system does not implement the EHR CRUD operations.

The users need to fill a form with their userid(created at enrollment), their role(either patient or auditor) and choose an action. There is provision for entering a record ID which is used only when the action is to test immutability.

Actions allowed are:

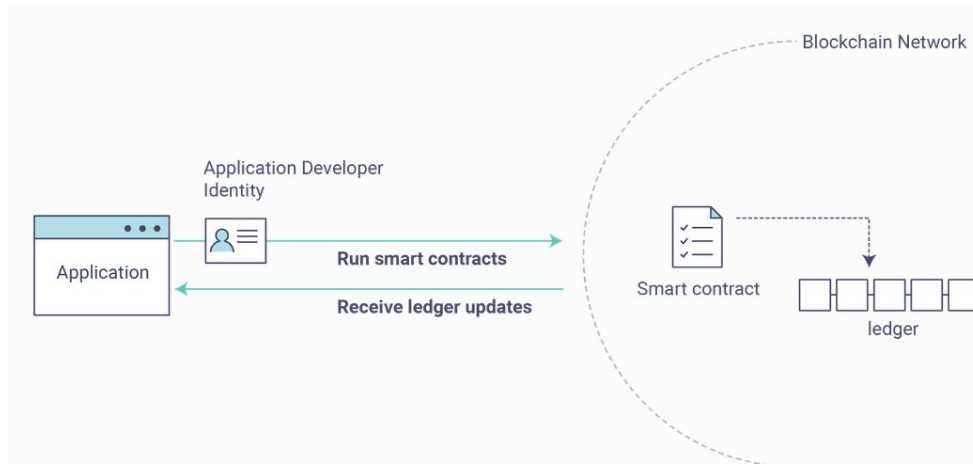
1. Query audit records(auditors when choosing query will see all the decrypted records in the ledger and patients will only see their own decrypted records )
2. View the raw contents of the ledger(all encrypted audit records).
3. Test immutability

Note that the query action will prompt the audit system to create more audit records of the type – query. Viewing the encrypted ledger does not generate audit data since it is not an operation but solely created

to show that ledger is encrypted and for the user to get access to a record ID to use in the immutability test.

## System architecture

This is the rough overview of the distributed architecture of the system.

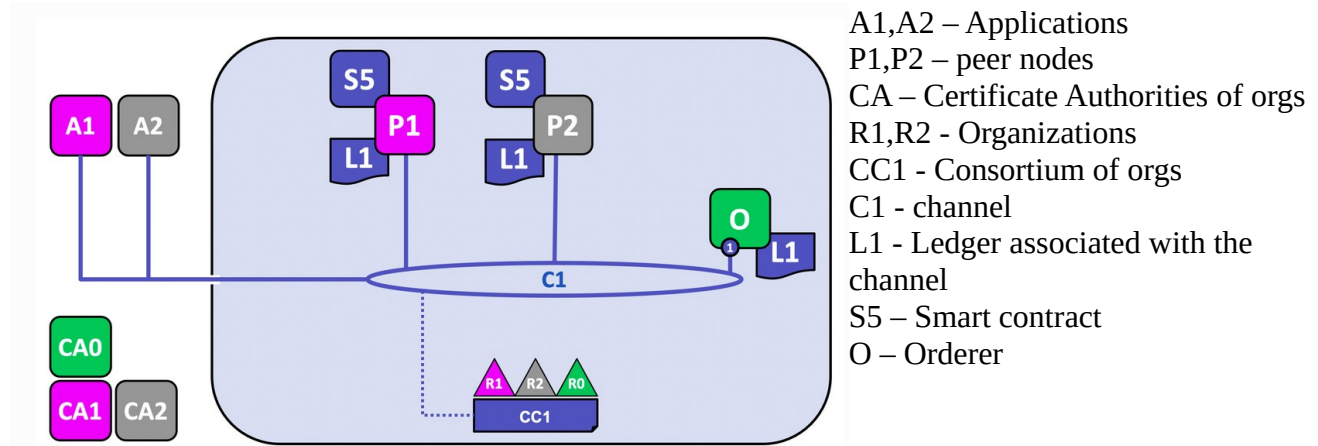


A blockchain network is set up first. It contains organizations. Each organization can register multiple peer nodes under it, and each peer can associate single or multiple user identities or admins identities. A channel is created that is associated with a single ledger. A peer who joins a channel is privy to that channel's blockchain or the ledger. Peers can join multiple channels, and organizations can form consortiums.

The channel/ledger is governed by a smart contract or chaincode. This smart contract definition contains API's that defines operations on the ledger. After channel is created, a smart contract has to be attached to it. Once peers join a channel, the smart contract is registered to the peers as well (i.e. they have to follow it to access the ledger). A number of orderer nodes can be attached to the channel (at least one) which perform all operations on the blockchain ledger, update state of shared ledger and broadcast to other peers with copy of the ledger etc. The orderer does this based on the smart contract.

An application acts as the interface to the blockchain network. Applications can be set up and taken down disjointly from the blockchain network. Applications are the only point-of-contact users have with the network. Applications have their own API's which internally refer to the smart contract API's for all network related operations. The application internally launches blockchain transaction which depending on the policy may or may not need to be endorsed by other entities, this transaction is then evaluated and if it conforms to the smart contract policy the transaction is successful.

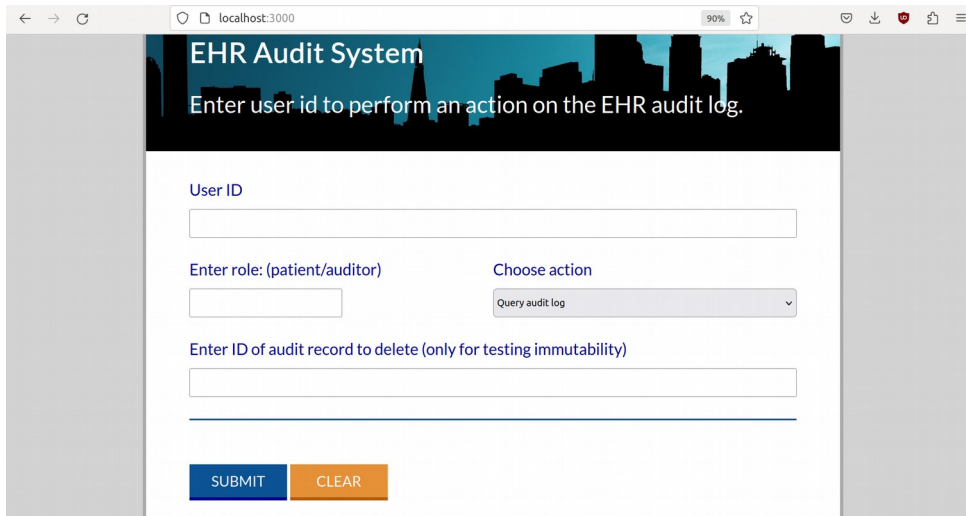
An overview of the blockchain network:



This prototype has the following architecture:

- 2 organizations: Org1 for patients and Org2 for auditors.
- Both orgs have one peer each.
- Org1 peer has enrolled 10 patients identities(created and stored their credentials) and 1 admin identity(who endorse blockchain transactions launched by users). Org2 peer has enrolled 3 auditors identities and 1 admin identity.
- There is one channel/ledger shared by all users, one smart contract and one orderer.
- There is one Javascript application run on the web browser which gives access to the blockchain network.

The front end:



### Audit Data format

The audit records have the below JSON format. These are the initial audit data entries that the ledger is populated with on system set up. The action\_type query is only added when audit records are queried and other types are supposed to be set during EHR CRUD operations.

```
const initdata = [  
  { patient_id: 'p1',  
    timestamp: '3/5/2022 18:5:48',  
    user_id: 'a1',  
    action_type: 'create',  
  },  
  { patient_id: 'p2',  
    timestamp: '5/5/2022 18:5:48',  
    user_id: 'a1',  
    action_type: 'delete',  
  },  
  { patient_id: 'p7',  
    timestamp: '8/5/2022 18:5:48',  
    user_id: 'a1',  
    action_type: 'change',  
  },  
  { patient_id: 'p3',  
    timestamp: '8/5/2022 18:5:48',  
    user_id: 'a1',  
    action_type: 'print',  
  }  
]
```

## Audit Record Format – as stored on the ledger

The record has a unique ID(SHA256 of user\_id concatenated with timestamp) which is stored in the clear and the entire JSON object with the actual audit data is encrypted and stored under the key record.

Encrypted audit record:

```
{
  "ID": "e33fbb71d9f2418f55d64190bb10670b4a8f1cd84e2b9691d4afd29866c0d908",
  "record": "c74e0cccabd2a07da5ff3c78d71edcd:3f0e657f0139195174937ffc2158d4765fbc0b78d19d3583e49ac480cdb4c53fd8b4623360dde5ea1849c9cc11ccbd60a1c31925f4b31394e17b2321f9595bd1a141392670449c3e849f6d23279b11d91c6513734d90b7cc"
}
```

Decrypted audit record:

```
[{"ID":"e33fbb71d9f2418f55d64190bb10670b4a8f1cd84e2b9691d4afd29866c0d908","record":
{"patient_id":"p1","timestamp":"3/5/2022 18:5:48","user_id":"a1","action_type":"create"}}]
```

## Cryptographic components

- Hyperledger has the blockchain technology in-built as well as all the cryptography components. The ledger is not implemented from scratch and no cryptographic components were created for implementing the blockchain itself.
- Fabric is a permissioned network, blockchain participants need a way to prove their identity to the rest of the network in order to transact on the network. PKI model has been used for this. CA servers have been used to create a certificate containing the public/private key pairs for a user upon enrollment(hardcoded for this prototype). There is a Fabric CA server present. The project has created 2 CA clients, one for each organization. The private-public key pair along with other credentials are created and stored for each user during enrollment in 2 separate wallets. The identities are used for various purposes such as maintaining history of transactions etc. Internally in Hyperledger Fabric.
- The JS library *crypto* has been used for:
  - The audit records are stored on the public ledger in encrypted format using AES-256 (CTR mode).
  - SHA-256 is used for hashing userid+timestamp to create unique records ID's.

## How system meets the outlined requirements

The system implements:-

- Privacy:** The system ensure that other patients(except auditors) can not query their records. The ledger stores audit in encrypted format using AES-256 CTR cipher. The record ID however is stored in the clear.  
The data that is in transit in the network is safeguarded using TLS internally in Fabric network. At rest, the data is stored on the blockchain which has provisions for maintaining integrity of the data.
- Identification and authorization:** Fabric has Membership Service Providers, one per organizations. The MSP contains a list of permissioned identities. The identity of the user is established by using a public/private key X.509 certificate. Hyperledger can provide access control using the CA certificates generated during enrollment. When asking to conduct a blockchain transaction, if the user lies and says that they are part of the auditor's organization i.e they are an auditor when they are actually a patient, or vice versa, the system can detect it by referring back to their credentials stored in the wallet and a connection to the system is not successful.
- Queries:** When patients query their data, all records which have their id as the patient id are extracted from the ledger, decrypted and then displayed. Auditors get all the decrypted records.

4. **Immutability:** Hyperledger is designed such that developers can create and fix the smart contract to control access to the ledger and the system. The record creation function in the smart contract does not allow any duplicates or the overwriting of an old record as during the creation the ID of the new asset/record is checked against existing one. This is unique since a SHA256 hash of userid+timestamp is used for the records ID's. So the system can not modify the audit records in error. The chaincode/smart contract does not have any API for updating the audit records. However, for the purpose of proving immutability, I have created an API to delete an audit record using the record ID. Hyperledger Fabric internally maintains history of audit records. The history is stored as a list of record values. The modification history of the record is printed once before deleting the record and once after it. When an audit record is deleted, the history add a new JSON object
5. **Decentralization:** Hyperledger Fabric has been used to create a distributed audit system

### System assumptions and limitations

The system does not have a provision for at-will enrollment of users(patients or auditors). At the beginning of the project, the enrollment of 10 patients, 3 auditors as part of 2 different organizations.

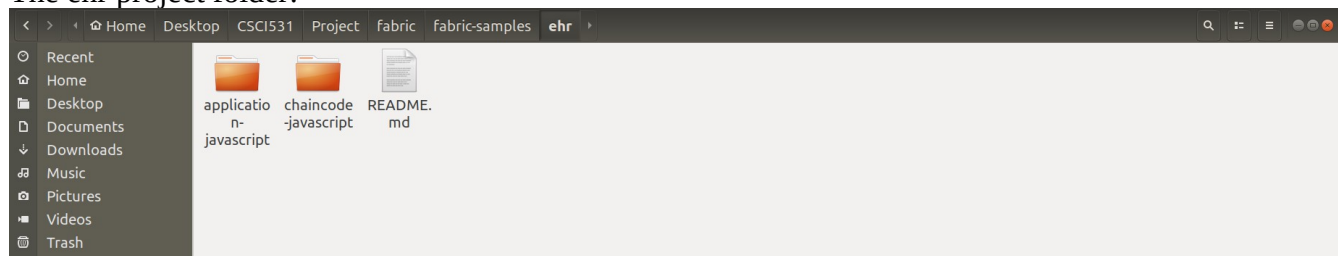
The system can be expanded as needed but for this prototype there are only 2 organizations and 2 peers. Each pair has multiple users registered but in a real system each organization will have a multiple peers and multiple users enrolled. Also, the auditors can be different organizations with multiple auditor identities in each. The system is scalable.

The ledger is populated with initial audit data as shown above. In a real system the encryption of audit records stored on the ledger(which is done by the Fabric application in this case) needs a random key every fixed number of days. But in this case the key is hardcoded in the code. Although, we use AES-256 in CTR mode, so every time the system or the application is relaunched, the initial data is same but the encryption will be different everytime time.

### Implementation

The project is created using Javascript. The audit system consists of the Fabric application and the blockchain network. The project explicitly requires coding the Fabric application to access the blockchain network and the smart contract or the chaincode to define the audit system. We can think of the blockchain network functioning as a back-end to our whole audit system.

The ehr project folder:



### 1. Fabric Application

This is a Node.js application (using Express framework). express.js framework creates a backend for the application, where the API's for and the front end is designed using HTML/CSS/Javascript.

The credentials of the patients and their admin i.e Organization 1 are stored in *application-javascript/wallet/org1* as *<user\_id>.id* and *admin.id* respectively. The credentials of the auditors and their admin i.e Organization 2 are stored in *application-javascript/wallet/org2*.

[illegible]

After this, the user can interact with the form to perform actions.

localhost:3000 90% ☆

Enter user id to perform an action on the EHR audit log.

User ID

Enter role: (patient/auditor) Choose action

Enter ID of audit record to delete (only for testing immutability)

```
[[{"ID": "e33fb71d9f2418f55d64190bb10670b4a8f1cd84e2b9691d4afd29866c0d908", "record": {"patient_id": "p1", "timestamp": "3/5/2022 18:5:48", "user_id": "a1", "action_type": "create"}}]]
```



## Querying for records: auditor

The screenshot shows a web browser window at localhost:3000. The application has a form with the following fields and controls:

- User ID:** A text input field containing 'a3'.
- Enter role: (patient/auditor):** A dropdown menu with 'auditor' selected.
- Choose action:** A dropdown menu with 'Query audit log' selected.
- Enter ID of audit record to delete (only for testing immutability):** A text input field containing '0'.
- Buttons:** 'SUBMIT' (blue) and 'CLEAR' (orange).

Below the form, a JSON array of audit records is displayed:

```
[{"ID": "08e98960466929fbc20ed4f35de3d38bec0c11d380fd7f628e687f0f5398d29e", "record": {"patient_id": "p1", "timestamp": "4/5/2023 21:8:36", "user_id": "p1", "action_type": "query"}}, {"ID": "364f03344c61342a2529611c1df14f57c00b31ee1313b4603a36a734d4f85927", "record": {"patient_id": "p7", "timestamp": "8/5/2022 18:5:48", "user_id": "a1", "action_type": "change"}}, {"ID": "5aedc1eb6e08aa3aa12008dea7674d76c5297fb721f3375062b8b854166aec31", "record": {"patient_id": "p3", "timestamp": "8/5/2022 18:5:48", "user_id": "a1", "action_type": "print"}}, {"ID": "8570b6a4cb72d0a68556d243b0b71e6feff936b56f7ede20ec84b9bbb562cd8", "record": {"patient_id": "p2", "timestamp": "5/5/2022 18:5:48", "user_id": "a1", "action_type": "delete"}}]
```

When patients or auditors query records, whichever patient's records were queried, the `patient_id` is used to store a new audit record with the `action_type` = query. For auditors, this will trigger a new audit record per every records that they have seen in the ledger.

## Querying for record – with mismatched authorization:

The screenshot shows the same web application interface as above, but with the role set to 'patient' and the action set to 'Query audit log'. Below the form, a blue error message is displayed:

Failed to Connect: Error: Identity not found in wallet: a3

The application will not connect when there is an authorization mismatch and this done my checking the `user_id` against the `mspId` stored in the wallet of the particular organization.

To test for immutability, user needs a valid record ID to delete which it can get by viewing the ledger in the raw encrypted state. Then it can enter the ID In the section given in the form. Internally the app is creating a transaction to get the modification of the audit record via `GetModificationHistory()` which uses Fabric funtion `getHistoryForKey()` on the blockchain, execute chaincode API `DeleteAsset()` to delete record and then another transaction to get the history of the record. The modified history shows 2



entries in the history i.e 2 version of the audit record. The recent one is a null string which means the record was deleted.

Viewing raw ledger:

The screenshot shows a web application interface for viewing the raw audit log. The title bar indicates the browser is at localhost:3000. The main heading is "Enter user id to perform an action on the EHR audit log." Below this, there are three input fields: "User ID" with the value "p2", "Enter role: (patient/auditor)" with the value "patient", and "Choose action" with a dropdown menu showing "View raw audit log (blockchain containing encrypted audit data)". There is also a field for "Enter ID of audit record to delete (only for testing immutability)" with the value "0". At the bottom, there are "SUBMIT" and "CLEAR" buttons. Below the form, a JSON array of audit records is displayed, showing multiple records with their IDs and encrypted data.

Immutability test:

The screenshot shows the same web application interface as before, but with the "Choose action" dropdown menu set to "Test Immutability (delete a record using record ID)". The "Enter ID of audit record to delete" field now contains a long hexadecimal string: "08e98960466929f20ed4f35de3d38bec0c11d380fd7f628e687f0f5398d29e". The "SUBMIT" and "CLEAR" buttons are still present. Below the form, a message indicates the modification history of the record before deleting Record 1, followed by a JSON array of audit records. The message states: "The most recent entry is a null string i.e. record has been deleted".

To view the records, the app calls GetAllAssets() from the smart contract and prints them as t is without decrypting the record value. To query particular records, the app process the audit records returned by GetAllAssets() and performs decryption using decrypt(), checks the pteint\_id against the patient\_id in the decrypted record if a patient is querying and prints only those records, or prints all decrypted records if an auditor is the one querying.

Every form submission results in a connection attempt to the network, which is closed after the action is complete. This means the audit system is a little slow, but is necessary to maintain access control.

## 2. Smart contract and Blockchain network

The ledger has 2 components: the world state which is the ledger with the audit records. Internally, LevelDB is used as database to store the copy of the public ledger on each peer. This DB instance is embedded on the same node but if we want we can configure the ledger to be stored on an external DB.

The second state is the ledger history which is maintained if the *enableHistoryDatabase* flag is True in *fabric-sample/config/core.yaml*. Make sure this is enabled or immutability test is not possible.

The smart code or the chaincode as Hyperledger Fabric calls it is stored in *ehr/chaincode-javascript/lib/assetTransfer.js*

The chaincode defines API's which are accessed by the Fabric app using Hyperledger's in-built API'. The call to chaincode API's are passed inside as a transaction using two functions *submitTransaction()* and *evaluateTransaction()*. The difference is *evaluateTransaction()* is used when the transaction does not make any change to the world state of the ledger.

All API's need the record ID and some additional parameters;

Chaincode has the following API's:

- *AssetExists()* - *AssetExists* returns true when asset with given ID exists in world state. Uses *getState()* internally.
- *CreateAsset()* - *CreateAsset* issues a new asset to the world state with given details. Checks if asset exists and if not calls *putState()*.
- *GetAllAssets()* - *GetAllAssets* returns all assets found in the world state. Internally performs a range query with empty string for *startKey* and *endKey* does an open-ended query of all assets in the chaincode namespace using *getStateByRange()*.
- *DeleteAsset()* - *DeleteAsset* deletes an given asset from the world state using *deleteState()*.
- *GetModificationHistory()* - *Get Modification history* of asset using *getHistoryForKey()* and iterate over the results to generate a JSON compliant list of JSON objects i.e each version of the audit record.

First, the block chain network has to be set up and established. And then the users can launch the Fabric API. Potentially, we can have many such apps as well.

### **Demo recording**

Shows each action, and explains the working with more cohesion as it is hard to describe the full implementation without visuals.

### **References**

- Charalampos Stamatellis, Pavlos Papadopoulos, Nikolaos Pitropakis, Sokratis Katsikas, William J Buchanan, A Privacy-Preserving Healthcare Framework Using Hyperledger Fabric, arXiv - CS - Cryptography and Security, 2020.
- <https://stackoverflow.com/questions/60369148/how-do-i-replace-deprecated-crypto-createcipher-in-node-js>
- <https://www.youtube.com/watch?v=rwKPXHUmks>
- <https://www.youtube.com/watch?v=iTV89Tqfmgk&list=PLsyeobzWxl7rXr9qxVZPbaoU7uUqP7iPM>