4.2 Adopted design steps for IPFS setup:

4.2.1 IPFS installation

The current project is developed on Mac OS Catalina

- Version: 10.15.6
- Processor: 1.4GHz Quad-Core Intel Core i5
- Memory: 8 GB 2133 MHz LPDDR3
- Graphics: Intel Iris Plus Graphics 645

To proceed with the development of the solution, installation of Visual Studio code editor, IPFS and Hyperledger are prerequisites. The following sections guide through the required installations on Mac OS.

System requirements for IPFS installation [46] requires 512MB of memory and base installation takes up to 12MB of memory with default maximum disk storage set to 10 GB. However, disk space utilization depends on how much data has been serving on the node. All the installation operation is done in the home directory.

4.2.2 IPFS Desktop Installation

IPFS desktop installation for Mac OS is mentioned in the IPFS desktop documentation [47]. This can be either installed through a downloadable desktop version or using a CLI interface. As the GUI interface gives an intuitive experience, the desktop application is downloaded from the site IPFS github relases and follows the instructions mentioned below. The desktop version comes with IPFS node, file manager, peer manager, and content explorer over GUI.

1. Download the latest available .dmg file from the IPFS github relases.
2. Open the ipfs-desktop.dmg file.
3. Drag the IPFS icon into the Applications folder:
4. Open your Applications folder and open the IPFS Desktop application.
5. You may get a warning saying IPFS Desktop.app can't be opened. Click Show in Finder.
6. Find IPFS Desktop.app in your Applications folder.
7. Hold down the control key, click IPFS Desktop.app, and click Open.
8. Click Open in the new window.
9. You can now find an IPFS icon in the status bar.

4.2.3 IPFS using CLI Installation

IPFS can be installed on M1-based Macs by using the darwin-arm64 binary instead of the amd64 binary listed in these instructions using CLI for Mac [48]. For other Operating System (OS) installation, refer to Official distributions [49]. This section can be skipped if desktop download is opted.

1. Download the Mac OS by giving the below command in CLI under home directory.

```
wget https://dist.ipfs.io/go-ipfs/v0.9.1/go-ipfs_v0.9.1
   _darwin-amd64.tar.gz
```

2. Run the following command to unzip the file:

```
tar -xvzf go-ipfs_v0.9.1_darwin-amd64.tar.gz
```

3. Move into the go-ipfs folder and run the install script:

```
cd go-ipfs
bash install.sh
```

4. Check that IPFS installed:

```
ipfs --version
```

5. Run the IPFS daemon:

```
ipfs daemon
```

6. Initialize the IPFS:

```
ipfs init
```

4.2.4 Creating IPFS Private Network:

The below steps are optional, but for development purposes, IPFS made to run on the local machine as a private node or the selected machines can be added to swarms to create a private network of nodes, this section can be skipped if it is needed to save the files on public domain IPFS.

S. V. Laar [50] suggested the instructions for deploying IPFS as a private network. Once the IPFS installation is done, we need to clear the bootstrap nodes that connect the public IPFS. To create a private network, we will use a swarm key. This swarm key will be referenced by all the nodes in this private network. The bootstrap node connects clients to other nodes available on the network. In a private network we cannot use the bootstrap nodes of the public IPFS network, so in this section we will replace the existing bootstrap with the IP address and peer identity of the local machine as bootstrap node.

1. IPFS is initialized in a hidden directory in your user home directory, run the following command in CLI.

```
IPFS_PATH=~/.ipfs ipfs init
```

2. To install the swarm key generator we use `\go get`, which uses git. If git is not installed, run the following command in CLI.

```
sudo apt-get install git
```

3. Run the following command to install the swarm key generator to create the swarm file.

```
go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-
   swarm-key-gen
```

4. Copy the generated swarm file to the .ipfs directory of all client nodes

```
./go/bin/ipfs-swarm-key-gen > ~/.ipfs/swarm.key
```

5. First, remove the default entries of bootstrap nodes from both the bootnode and the client node. Run the following command in CLI.

```
IPFS_PATH=~/.ipfs ipfs bootstrap rm --all
```

6. Check the result to see the bootstrap key value is empty with help of below command.

```
IPFS_PATH=~/.ipfs ipfs config show
```

7. Get the peer ID of the node by issuing the following command.

```
IPFS_PATH=~/.ipfs ipfs config show | grep "PeerID"
```

8. Get the IP address of Mac machine by clicking icon on top left corner, then choose *'System preferences -> Sharing -> Remote management'*. This will display the IP address of the machine.
9. Now add the IP address and the Peer Identity (hash address) of Mac machine.
10. Run the following command to add the local machine to private network.

```
IPFS_PATH=~/.ipfs ipfs bootstrap add /ip4/<ip address of
   bootnode>/tcp/4001/ipfs/<peer identity hash of
   bootnode>
```

11. This completes the set up of private network, test this network by using an environment variable to make sure the configuration done properly. Else, IPFS network and daemons will fail. Use the environment variable "LIBP2P_FORCE_PNET" and to start the IPFS node with the following command.

```
export LIBP2P_FORCE_PNET=1
IPFS_PATH=~/.ipfs ipfs daemon &
```

Setting LIBP2P_FORCE_PNET=1 means to force the node to be private. If no private network is configured, the daemon will fail to start. Output from the above command should state `\Swarm is limited to private network of peers with the swarm key`.

4.3 Adopted design steps for Hyperledger setup:

4.3.1 Hyperledger Prerequisites Installation:

Before setting up the Hyperledger, ensure the below prerequisites are installed on the system. The current project has been developed on the Mac OS, hence the steps mentioned as per the system OS. If the OS is different, following the source documentation [51] accordingly to get the essentials installed.

1. Download the latest version of Visual Studio code editor for your system and install it. This is chosen as it have good interface for CLI and editor.
2. Download the latest version of git and install, if it is not already installed.
3. Download the latest version of the cURL tool and install, if it is not already installed.
4. Download the Docker Docker version 17.06.2-ce or greater is required for MacOSX, *nix, or Windows 10.
5. The version of the Docker installed can be checked with the following command from a terminal prompt.

```
docker --version
```

6. Make sure the docker daemon is running by using the following command.

```
sudo systemctl start docker
```

7. This command is optional, If you want the docker daemon to start when the system starts.

```
sudo systemctl enable docker
```

8. Add your user to the docker group.

```
sudo usermod -a -G docker <username>
```

4.3.2 Installing Hyperledger Fabric:

1. Below installing instructions are explained in the Hyperledger Fabric Documentation, Follow the installation documentation [52] need to be done on different OS.
2. As the development has been done on Mac OS with help of docker, the ledger installers should be downloaded to the location under /Users, /Volumes, /private, or /tmp. /Users file location (home directory) chosen for installer location in the current project.
3. Go to /Users location on your machine using terminal window where you want to place the fabric-samples repository (by default CLI points to the home directory).
4. Check out the appropriate version tag, if need to install a specific version. Use the below cURL command to get the latest fabric version which results in the following steps.

```
curl -sSL https://bit.ly/2ysbOFE | bash -s
```

If you want a specific release, pass a version identifier for Fabric and Fabric-CA docker images.

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- <
   fabric_version> <fabric-ca_version>
```

5. It clones the hyperledger/fabric-samples repository.
6. Installs the Hyperledger Fabric platform-specific binaries and config files for the version specified into the /bin and /config directories inside fabric-samples folder.
7. Downloads the Hyperledger Fabric docker images for the version specified
8. If an error raises running the above curl command, it might be of an old version of curl that does not handle redirects or an unsupported environment.
9. Add the PATH environment variable so that bin files can be picked up without fully qualifying the path to each binary.
   e.g.

```
export PATH=<path to download location>/bin:$PATH
```

10. Finally, the script will download the Hyperledger Fabric docker images from Docker Hub into your local Docker registry and tag them as 'latest'.
11. Hyperledger Fabric offers several SDKs to support developing applications in various programming languages. SDKs are available for Node.js and Java. In the current project, Node.js is being used for development.

4.3.3 Installation Test:

The steps till above section should help the proper installation of Hyperledger Fabric. The following steps are to test the Hyperledger installation by using the Command line interface (CLI) program to ensure no issues with the downloaded version.

1. The following instructions are explained in *Using test network*, follow the documentation [53] for in-depth information.
2. To ensure that installation has been done properly. Navigate to the *test-network* directory by using the following command from the home directory.

```
cd fabric-samples/test-network
```

3. In this directory, you can find an annotated script, network.sh, that stands up a Fabric network using the Docker images on your local machine. You can run the below command to print the script help text.

```
./network.sh -h
```

4. From inside the *test-network* directory, run the following command to remove any containers or artifacts from any previous runs.

```
./network.sh down
```

5. To bring up the network issue the following command from *fabric-samples/test-network* directory.

```
./network.sh up
```

6. The above command creates a Fabric network that consists of two peer nodes, one ordering node. No channel is created yet.
7. Run the following command to list all of Docker containers that are running on your machine.

```
docker ps -a
```

8. The test network includes two peer organizations, Org1 and Org2. It also includes a single orderer organization that maintains the ordering service of the network.
9. Peers are the fundamental components of any Fabric network. They store the blockchain ledger and validate transactions before they are committed to the ledger. Peers run the smart contracts that contain the business logic that is used to manage the assets on the blockchain ledger.
10. In the test network, each organization operates one peer each, peer0.org 1.example.com and peer0.org2.example.com
11. An ordering service allows peers to focus on validating transactions and committing them to the ledger. After ordering nodes receive endorsed transactions from clients, they come to a consensus on the order of transactions and then add them to blocks.
12. The following script is used to create a Fabric channel for transactions between Org1 and Org2. Channels are a private layer of communication between specific network members.
13. Run the following command to create a channel with the default name of *'mychannel'*, if no name is specified explicitly.

```
./network.sh createChannel
```

14. In the future, both the network and channel can be created in a single step using the below command.

```
./network.sh up createChannel
```

15. Custom channel name can be used with following command. Current application uses the default channel name *'mychannel'*.

```
./network.sh createChannel -c channel1
```

16. After channel creation, smart contracts can be invoked to interact with the channel ledger. Applications run by members of the network can invoke smart contracts to create assets on the ledger, as well as change and transfer those assets.
17. To start a chaincode on the channel use the following command.

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-
   basic/chaincode-go -ccl go
```

18. The *'deployCC'* subcommand will install the asset-transfer (basic) chaincode on peer0.org1.example.com and peer0.org2.example.com and then deploy the chaincode on the channel specified using the channel flag (or mychannel if no channel is specified). The language flag, *-l*, to install the Go, typescript or javascript versions of the chaincode. The basic chaincode can be found in the of the *fabric-samples/asset-transfer-basic folder* directory.

4.3.4 Interacting with the network:

After bringing up the test network, now the peer CLI can be used to interact with your network using the bin files. In-depth instructions are mentioned in interacting with the network [54].

1. Peer CLI allows you to invoke deployed smart contracts, update channels, or install and deploy new smart contracts from the CLI.
2. Make sure that CLI operations are from the *test-network* directory. If the installation is done properly, the peer binaries will be in the *bin* folder of the *fabric-samples* repository.
3. Use the following command to add those binaries to your CLI Path.

```
export PATH=${PWD}/../bin:$PATH
```

4. Set the FABRIC_CFG_PATH to point to the core.yaml file in the *fabric-samples* repository.

```
export FABRIC_CFG_PATH=$PWD/../config/
```

5. You can now set the environment variables that allow you to operate the peer CLI as Org1.

```
# Environment variables for Org1
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
   peerOrganizations/org1.example.com/peers/peer0.org1.
   example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
   peerOrganizations/org1.example.com/users/Admin@org1.
   example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
```

6. If CLI command *./network.sh deployCC -ccl go* is used to install and start the asset-transfer (basic) chaincode, it will invoke the *InitLedger* function of the (Go) chaincode to put an initial list of assets on the ledger. Similarly, the respective chaincode function invokes based on the *ccl* parameter.
7. Run the following command to initialize the ledger with assets.

```
peer chaincode invoke -o localhost:7050 --
   ordererTLSHostnameOverride orderer.example.com --tls
   --cafile ${PWD}/organizations/ordererOrganizations/
   example.com/orderers/orderer.example.com/msp/
   tlscacerts/tlsca.example.com-cert.pem -C mychannel -n
   basic --peerAddresses localhost:7051 --
   tlsRootCertFiles ${PWD}/organizations/
   peerOrganizations/org1.example.com/peers/peer0.org1.
   example.com/tls/ca.crt --peerAddresses localhost:9051
   --tlsRootCertFiles ${PWD}/organizations/
   peerOrganizations/org2.example.com/peers/peer0.org2.
   example.com/tls/ca.crt -c '{"function":"InitLedger","
   Args":[]}'
```

8. If successful, should give the output as below.

```
-> INFO 001 Chaincode invoke successful. result: status
   :200
```

9. Now query the ledger from your CLI. Run the following command to get the list of assets that were added to your channel ledger.

```
peer chaincode query -C mychannel -n basic -c '{"Args":["
   GetAllAssets"]}'
```

10. If successful, it should give the following output.

```
[
{"ID": "asset1", "color": "blue", "size": 5, "owner": "
   Tomoko", "appraisedValue": 300},
{"ID": "asset2", "color": "red", "size": 5, "owner": "
   Brad", "appraisedValue": 400},
{"ID": "asset3", "color": "green", "size": 10, "owner": "
   Jin Soo", "appraisedValue": 500},
{"ID": "asset4", "color": "yellow", "size": 10, "owner":
   "Max", "appraisedValue": 600},
{"ID": "asset5", "color": "black", "size": 15, "owner": "
   Adriana", "appraisedValue": 700},
{"ID": "asset6", "color": "white", "size": 15, "owner": "
   Michel", "appraisedValue": 800}
```

```
]
```

11. Use the following command to change the owner of an asset on the ledger by invoking the asset-transfer (basic) chaincode.

```
peer chaincode invoke -o localhost:7050 --
   ordererTLSHostnameOverride orderer.example.com --tls
   --cafile ${PWD}/organizations/ordererOrganizations/
   example.com/orderers/orderer.example.com/msp/
   tlscacerts/tlsca.example.com-cert.pem -C mychannel -n
   basic --peerAddresses localhost:7051 --
   tlsRootCertFiles ${PWD}/organizations/
   peerOrganizations/org1.example.com/peers/peer0.org1.
   example.com/tls/ca.crt --peerAddresses localhost:9051
   --tlsRootCertFiles ${PWD}/organizations/
   peerOrganizations/org2.example.com/peers/peer0.org2.
   example.com/tls/ca.crt -c '{"function":"TransferAsset
   ","Args":["asset6","Christopher"]}'
```

12. If the command is successful, you should see the following response.

```
-> INFO 001 Chaincode invoke successful. result: status
   :200
```

13. Because the endorsement policy for the asset-transfer (basic) chaincode requires the transaction to be signed by Org1 and Org2, the chaincode invoke command needs to target both peer0.org1.example.com and peer0 .org2.example.com using the *–peerAddresses* flag. Because TLS is enabled for the network, the command also needs to reference the TLS certificate for each peer using the *–tlsRootCertFiles* flag.
14. After the chaincode is invoked, use another query to see how the invoke changed the assets on the blockchain ledger. Since the Org1 peer is queried, now the chaincode can be queried from Org2 peer. Set the following environment variables to operate as Org2.

```
# Environment variables for Org2

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
   peerOrganizations/org2.example.com/peers/peer0.org2.
   example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
   peerOrganizations/org2.example.com/users/Admin@org2.
   example.com/msp
export CORE_PEER_ADDRESS=localhost:9051
```

15. Now query the asset-transfer (basic) chaincode running on peer0.org2.example.com.

```
peer chaincode query -C mychannel -n basic -c '{"Args":["
   ReadAsset","asset6"]}'
```

16. The result will show that *'asset6'* was transferred to Christopher.

```
{"ID":"asset6","color":"white","size":15,"owner":"
   Christopher","appraisedValue":800}
```

17. When working with the test network is finished, the network can be brought down with the following command.

```
./network.sh down
```

18. The command will stop and remove the node and chaincode containers, delete the organization crypto material, and remove the chaincode images from your Docker registry. The command also removes the channel artifacts and docker volumes from previous runs.
19. If the network needs to use Fabric CAs, first run the following command while bringing up the network in the future.

```
./network.sh up -ca
```

4.4 Steps for Engineered solution design:

With successful Hyperledger installation and IPFS set up, this section guides, how to build the web application based on Node.js that helps interact with Hyperledger, IPFS. CLI commands in the before section helped in familiarising how to query the chaincode. As it is a basic chaincode, it doesn't have features of querying the asset history. So, now the work will be focussed on the application template available in *asset-transfer-ledger-queries* folder in *fabric-samples*. For node application to run, it is required to install *npm* and the below steps help to build the web application.

1. Ensure the network is down from any previous trails.
2. Open the Visual Studio code editor.
3. First, navigate to the *asset-transfer-ledger-queries/application-javascript* folder in *fabric-samples* through editor file option menu. Open the file *app.js* and familiarize what kind of ledger accessing functionalities it can perform.

4. We are directly building the web application without the need to run the app from the terminal. Edit the *app.js* with the code from appendix- 6.2.1 and save the file.
5. This is the main file, that drives the web application that helps to upload the image files to IPFS and interact with Hyperledger.
6. Open the terminal window in Visual Studio code editor, should pick up the current directory as working directory else navigate to the working path *asset-transfer-ledger-queries/application-javascript* in CLI and install *npm* packages in the path and initialize it with the below commands (only for the first time setup).

```
npm install
```

7. The above should take few minutes to install the node modules in the current working directory, after completion of the above run the below command, it initializes the dependencies specified in *package.json* in the working directory.

```
npm init
```

8. Install the following dependencies individually in the working directory that helps to run the web application. These new dependencies should be reflected in the *package.json* file in the working path.

```
# EXECUTE THEM INDEPENDENTLY AND WAIT TILL EACH
   INSTALLATION IS DONE
npm install body-parser
npm install child_process
npm install ejs
npm install express
npm install express-fileupload
npm install fabric-common
npm install fabric-network
npm install formidable
npm install http
npm install ipfs
npm install ipfs-api
npm install ipfs-http-client
npm install protobufjs
```

9. After the required module installation, check the *package.json* file in the working directory should reflect these dependencies.
10. Now, navigate to the *asset-transfer-ledger-queries/chaincode-javascript/lib* folder using file explorer. Ledger chaincode file is available here with the name *asset_transfer_ledger_chaincode.js*. Open it with using code editor.

11. Familiarize with chaincode functionalities it offers and copy the content from appendix-6.2.2 that customizes image ownership data to be stored on the Hyperledger.
12. Now navigate to index.js file in *asset-transfer-ledger-queries/chaincode-javascript* using file explorer and make sure that script file is pointing to *./lib/asset_transfer_ledger_chaincode.js*.
13. The application uses couch DB as the data store for ledger transactions and current data state. Hence we need to raise the network passing the database parameter option.
14. To run the application we need to go to the command line terminal in the Visual Studio code editor and navigate to the *fabric-samples/test-network* from working directory *asset-transfer-ledger-queries/application-javascript* with help of below commands.

```
cd ..
cd ..
cd test-network/
```

15. Issue the following command line that will raise the raise Hyperledger network from clean base, creates *'mychannel'*, sets the network to use Certification Authority to build the crypto material for the channel users, and signifies *'couchdb'* is used to store the state database.

```
./network.sh down
./network.sh up createChannel -c mychannel -ca -s couchdb
```

16. Wait for few minutes until the above operations are done and will be waiting for the next command line input.
17. After creating the channel, now deploy the *'ledger'* chaincode with the help of chaincode edited in before section. For this experiment javascript is used as chaincode language.

```
./network.sh deployCC -ccn ledger -ccp ../asset-transfer-
   ledger-queries/chaincode-javascript/ -ccl javascript
```

18. Wait for few mintues until the chaincode package is deployed and CLI waits for the next input command.
19. For understanding what happened behind the above issued command lines:
    - Creates genesis block in path *test-network/system-genesis-block/*.
    - Creates configuration folder for peers and orderer organisations in *test-network//organizations/*.
    - Creates packaged chaincode *ledger.tar.gz* in *test-network* folder.
20. Now navigate to the *asset-transfer-ledger-queries/application-javascript/app.js* path in the CLI and run the app by issuing the following command.

```
# EXECUTE THEM INDIVIDUALLY TO UNDERSTAND NAVIGATION
cd ..
cd asset-transfer-ledger-queries/application-javascript/
ls
node app.js
```

21. Open your desired browser, navigate to the URL http://localhost:3000/ on your first run as it will initialize the default users to channel and creates a default entry on a ledger.
22. Behind the scenes after accessing that URL, it creates the wallet folder that stores the admin and users' crypto material which authenticates to access the network in the *asset-transfer-ledger-queries/application-javascript/* path.
23. Conceptual diagram for application is shown in Fig.:4.1.
24. Describing the menu items from the web application, the first tab *Upload to IPFS* helps to upload the files and give some user information and click upload (Fig:5.1,5.3), depends upon the IPFS operation mode, files will be saved locally or public.
25. After successful upload, it will show the uploaded information (Fig.:5.4).
26. *Load all* (Fig.:5.2,5.5,5.7) tab shows all the asset information stored on the ledger. Upon clicking the individual item, expands the trails of transactions that files have been through.
27. It has an *edit* action button that lets to update the ownership information and update the ledger (Fig.:5.6).
28. *Search asset* tab helps to find the file information by accepting file input which will query the ledger with its file hash and displays the current information it holds (Fig.:5.8).
29. *Register user* (Fig.:5.9) and *Registered users* (Fig.:5.10) are for the generic purpose that lets you add users to the channel. All the executed transactions utilize the default user ID created while accessing the app.
30. If interaction with the application is done. Exit the node process by command `\ctrl+c` in the terminal window of Visual Studio code editor.
31. If it needs to bring down the network entirely, halt the node process. Navigate to *fabric-samples/test-network/* folder in CLI using the following command.

```
cd ..
cd ..
cd test-network
```

32. Execute the following command on terminal window of Visual Studio code editor and wait for a few minutes. This brings down the network and deletes the docker containers.

```
./network.sh down
```

33. Delete the wallet folder generated in the path *fabric-samples/asset-transfer-ledger-queries/application-javascript/* using file explorer, else it will give an error as *'access denied'* upon the next time network initiation while accessing the http://localhost:3000/.
34. If deleting the wallet folder from the previous run is missed, stop the node process and delete the folder now and again start the node application by accessing URL http://localhost:3000/ from the browser.
35. *'access denied'* error can be rectified with the above step and creates wallet folder again with new crypto material.

[44] M. Y. Jabarulla and H. Lee, *'Blockchain-Based Distributed Patient-Centric Image Management System,'* 2020. DOI: 10.3390/app11010196 Available: https://arxiv.org/abs/2003.08054v4.

[45] Ichikawa, D., Kashiyama, M., Ueno, T. (2017). *'Tamper-Resistant Mobile Health Using Blockchain Technology.'* JMIR mHealth and uHealth, 5(7), e111. https://doi.org/10.2196/mhealth.7938. Available : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5550736/.

[46] IPFS Docs. *'System requirements'*. Available: https://docs.ipfs.io/install/command-line/#system-requirements.

[47] IPFS Docs. *'IPFS desktop installation - MacOS'*. Available: https://docs.ipfs.io/install/ipfs-desktop/#macos.

[48] IPFS Docs. *'IPFS Command-line - MacOS'*. Available: https://docs.ipfs.io/install/command-line/#macos.

[49] IPFS Docs. *'IPFS Official distributions'*. Available: https://docs.ipfs.io/install/command-line/#official-distributions.

[50] S. V. Laar. January 3, 2020. *'Deploy a private IPFS network on Ubuntu in 5 steps'*. Available: https://medium.com/@s_van_laar/deploy-a-private-ipfs-network-on-ubuntu-in-5-steps-5aad95f7261b .

[51] Hyperledger Fabric. *'Prerequisites'*. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html .

[52] Hyperledger Fabric. *'Windows extras'*. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html#windows-extras .

[53] Hyperledger Fabric. *'Using the Fabric test network'*. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/test_network.html .

[54] Hyperledger Fabric. *'Interacting with the network'*. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/test_network.html#interacting-with-the-network.