

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Fabric SDK for Node.js

This module will cover building front-end interface to interact with the smart contract. This module will enable complete understanding of the development of an entire application on the Hyperledger Fabric framework. Some prerequisites:

- ◆ Node.js
- ◆ NPM tool
- ◆ YAML
- ◆ HTML

Introduction to the Application:

In the prior module, the Certification Network was built with 3 organizations with 2 peers each, a CLI and a chaincode container and a channel named *certification-channel*. A chaincode was written with certain logic using Node.js and it was installed on the peers and instantiated and made available on the channel. We then used *fabric-contract-api* to access the ledgers on the peer and perform the functions of the smart contract.

This smart contract is now available on the channel and will be used by the various client applications. Each of the organizations can have applications of their own to interface with the network. For example, IIT would like to have a web/mobile application which would enable them to enroll students and issue certificates. MHRD being a central organization would like to have more control over the entire certification process and may have admin capabilities. Similarly, upGrad would have applications that provide capability to verify certificates instantly.

As a simple use case, in this module, MHRD organization has been modeled to have the ability to create student, issue certificate and also verify the validity of the certificates. To enable this, the front end application would need API's to interact with the Fabric network. This application will first connect to the network, connect the correct channel, find the right chaincode/smart contract and invoke the functions/perform transactions or query the ledgers. This is made possible by Fabric SDK using Node.js.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

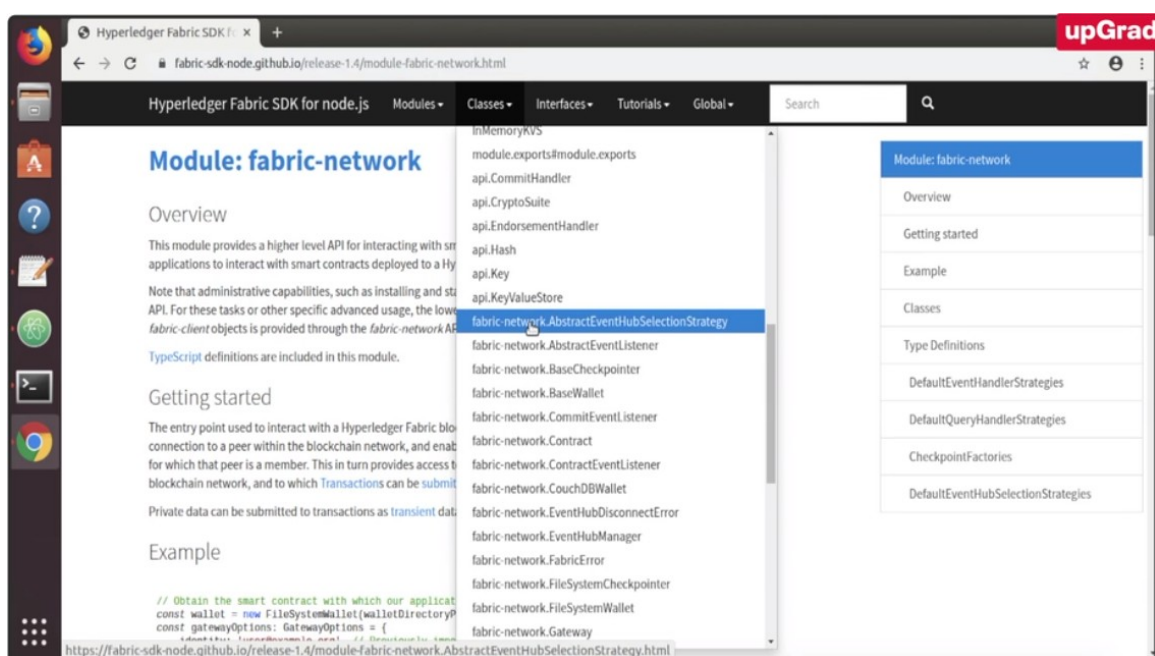
APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Fabric Node SDK:

This *fabric-sdk-node* kit provides the functionalities to the client application to interact with the Fabric network. It has four major modules:

- fabric-network:
 - : submit transactions
 - : query ledgers
- api
 - : provides low level functions
- fabric-client:
 - : provides capabilities for management interface like view, control, modify network configurations like peers, channels etc
- fabric-ca-client:
 - : provides functionality to access the CA (Certificate Authority)

Please browse the [www](https://fabric-sdk-node.github.io/release-1.4/module-fabric-network.html) for the latest website link to access the Fabric SDK documentation. This SDK and also the documentation is rapidly evolving and provides a wiki page on the classes available as a part of these API's.



HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

fabric-network:

The four functions 1) Create Student, 2) Get Student, 3) Issue Certificate, 4) Verify Certificate needs to be accessed by the client application that we will build. Hence the client application will need to access the chaincode that has been instantiated on the channel. The fabric-network SDK has the API's to invoke the functions in the smart contract.

Following are the steps to access the chaincode and submit a transaction:

- Define a wallet:

Create a Wallet directory within which the identities that are required to connect to the network. The instance of *FileSystemWallet* class will be used to generate the identities with the wallet and the directory path is provided to this as a parameter.

- Define connection parameters:

Parameters which are required to connect to the network like the URL, TLS related PEM certificates, timeout etc. All of these are defined as a single YAML file called CCP (Common Connection Profile). This also defines the network topology such as the network peers, endorsers and committers.

- Build a gateway:

Gateway is an object/class provided by the SDK which gives the ability to create and maintain a connection to the Fabric network. The instance of the *Gateway* class provides various methods to connect to the network.

- Connect:

The `Gateway.connect()` method allows connection to the network using the identity of the user and the CCP defined before.

- Access the channel:

Use the `Gateway.getNetwork()` method to get an instance of the channel by providing the channel name.

- Access the chaincode:

Use the `getContract` method to get an instance of the chaincode.

- Submit transaction:

Use `createTransaction()` method to invoke one of the functions in the contract.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Common Connection Profile

This module covers the following:

- 1) The architecture of the application for the certification network
- 2) Common connection profile

Application Architecture:

The application will interface with the network using the gateway. The gateway will provide the functionalities using which the application can access the channel, the chaincode and invoke the transactions. The application will inherit the Fabric SDK for this purpose. We will build independent node modules to provide the core functionalities such as creating a student, issuing certificate, get the details of the student and verifying the certificate. These individual modules will be run from the terminal by the different peers. To enable any HTML based web page access these modules, they will be exported as server side API's.

Following is the structure of this client application:

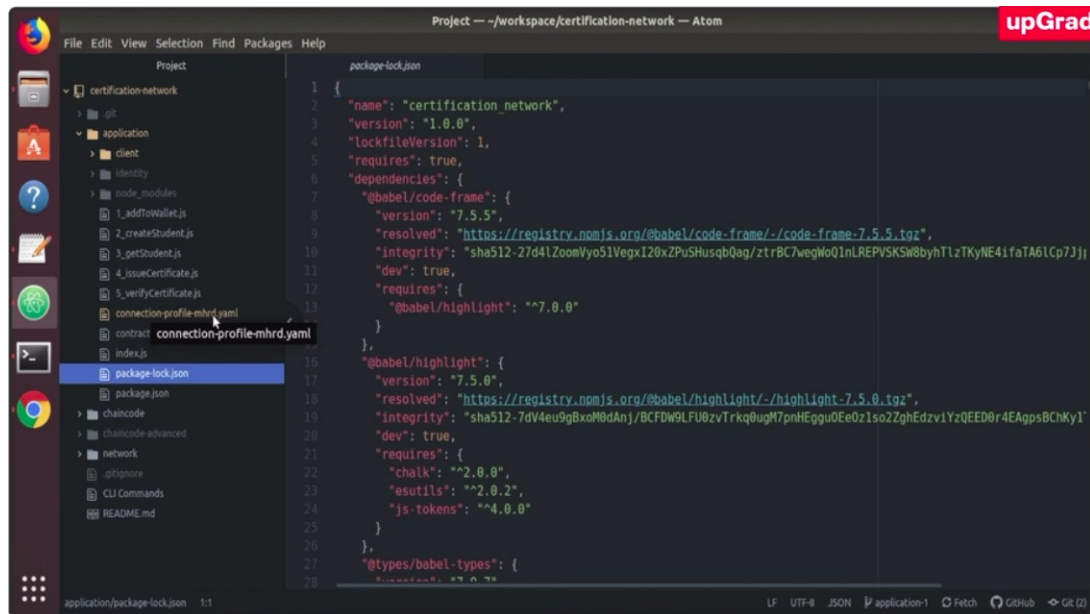
- individual node modules
 - : these provide the functionalities that the end users will perform
- server
 - : this will expose the modules to the front end UI
- client UI
 - : HTML page which can access the API's exposed by the server

Besides the node modules, the application project folder will have the *client* folder which will have the files used for the HTML page. The *identity* folder is the wallet which will have all the identities of the users of this client application. The *package.json* file defines the parameters for the application. The *index.js* file is the main landing file for the application. In our case, we have five different node module files for various functionalities. The connection profile is the YAML file which has the configurations and properties for the MHRD organization.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC



Common Connection Profile - I:

CCP (Common Connection Profile) is a YAML file which will list down all the parameters that the client application will need to connect to the Fabric network. Following are the parameters:

- name:
this is the name for the profile
- x-type:
this is the version of Fabric on which the peers and orderers are running. Here, only the major version number is mentioned
- description:
description for this connection profile
- version:
version of the YAML file
- channels:
list of channels in the network and the entities which are part of the channel. In our case, it is the *certificationchannel*. The entities would be the orderer and the peers of each organization. There will be four properties for each of these entities as below:

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

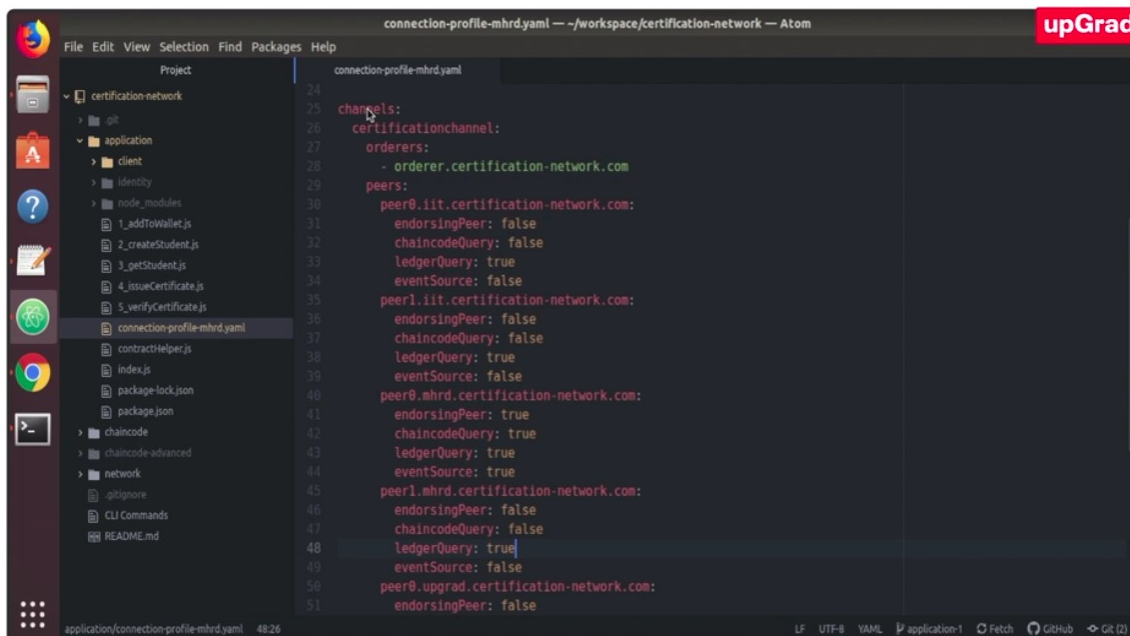
APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

: endorsingPeer: denotes if the peer is an endorsing peer. Since this example is written for the connection profile of the MHRD organization, this property is set to true only the peer0 of the MHRD. This means that any transaction initiated by the client application using this CCP will be sent only to peer0 of MHRD for endorsement

: chaincodeQuery: if set to true, this will allow the transactions to be queried on this specific peer

: ledgerQuery: this is set to enable the client application fetch the state of the assets on the network which is a part of the ledgers on the peers. This is set to true for all the peers in this case since all the peers are committers

: eventSource: whenever a chaincode triggers an event, the client application can subscribe to those events. This property will define if the peer is a source of events



```
connection-profile-mhrd.yaml
24
25 channels:
26   certificationchannel:
27     orderers:
28       - orderer.certification-network.com
29   peers:
30     peer0.iit.certification-network.com:
31       endorsingPeer: false
32       chaincodeQuery: false
33       ledgerQuery: true
34       eventSource: false
35     peer1.iit.certification-network.com:
36       endorsingPeer: false
37       chaincodeQuery: false
38       ledgerQuery: true
39       eventSource: false
40     peer0.mhrd.certification-network.com:
41       endorsingPeer: true
42       chaincodeQuery: true
43       ledgerQuery: true
44       eventSource: true
45     peer1.mhrd.certification-network.com:
46       endorsingPeer: false
47       chaincodeQuery: false
48       ledgerQuery: true
49       eventSource: false
50     peer0.upgrad.certification-network.com:
51       endorsingPeer: false
```

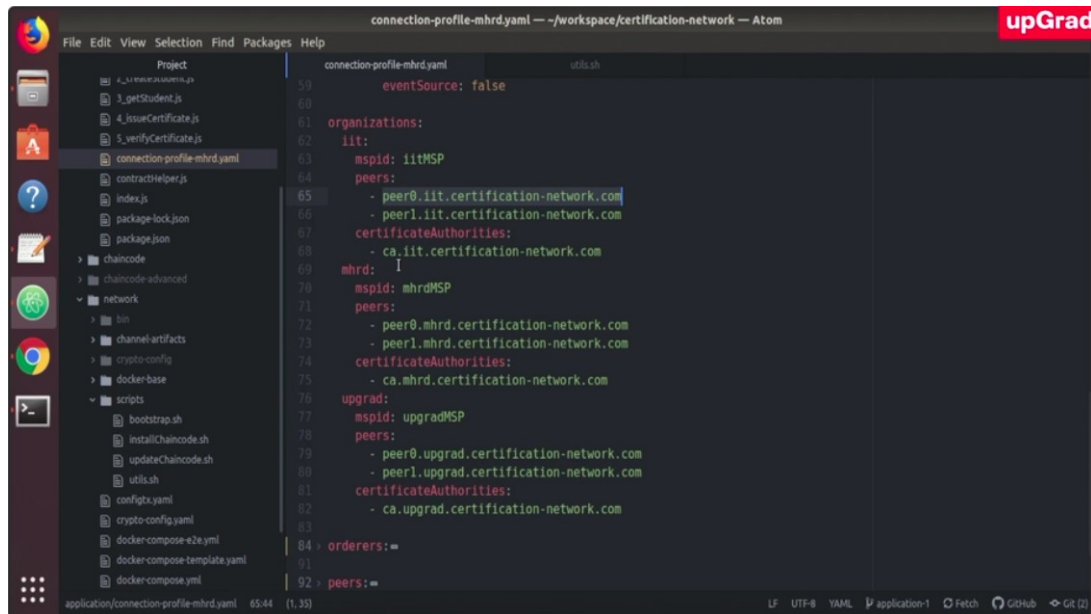
- organizations:

this property defines the organizations within the network. Within this, the mspId, the address of the peers and the CA's are defined.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

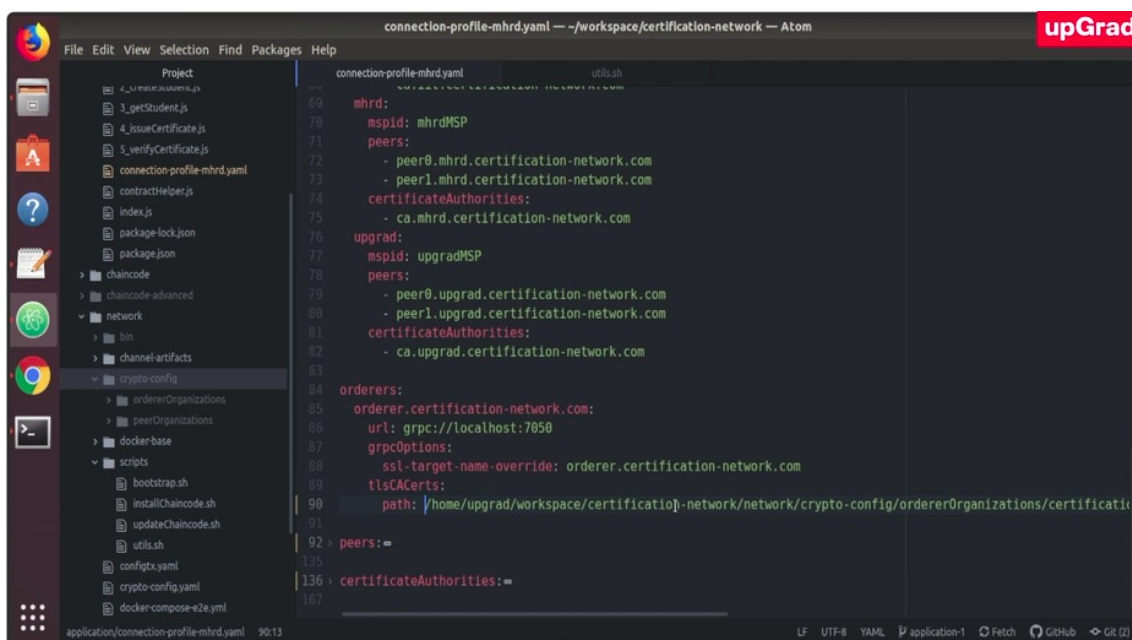


```
connection-profile-mhrd.yaml
eventSource: false
organizations:
  iit:
    mspid: iitMSP
    peers:
      - peer0.iit.certification-network.com
      - peer1.iit.certification-network.com
    certificateAuthorities:
      - ca.iit.certification-network.com
  mhrd:
    mspid: mhrdMSP
    peers:
      - peer0.mhrd.certification-network.com
      - peer1.mhrd.certification-network.com
    certificateAuthorities:
      - ca.mhrd.certification-network.com
  upgrad:
    mspid: upgradMSP
    peers:
      - peer0.upgrad.certification-network.com
      - peer1.upgrad.certification-network.com
    certificateAuthorities:
      - ca.upgrad.certification-network.com
84 orderers:
91
92 peers:

```

- orderers:

defines the orderers in the network. As a part of this property, the url, grpcOptions and the tlsCACerts are defined. *grpc* is the protocol used to connect to the orderer. If TLS was enabled in the network, the *grpcs* will be used. The certificates generated while building the network will be reused and hence it is important to specify the path of the certificates correctly.



```
connection-profile-mhrd.yaml
mhrd:
  mspid: mhrdMSP
  peers:
    - peer0.mhrd.certification-network.com
    - peer1.mhrd.certification-network.com
  certificateAuthorities:
    - ca.mhrd.certification-network.com
upgrad:
  mspid: upgradMSP
  peers:
    - peer0.upgrad.certification-network.com
    - peer1.upgrad.certification-network.com
  certificateAuthorities:
    - ca.upgrad.certification-network.com
84 orderers:
85   orderer.certification-network.com:
86     url: grpc://localhost:7050
87     grpcOptions:
88       ssl-target-name-override: orderer.certification-network.com
89     tlsCACerts:
90       path: /home/upgrad/workspace/certification-network/network/crypto-config/ordererOrganizations/certificati
91
92 peers:
135
136 certificateAuthorities:
167

```


HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

What is the difference between the properties ‘endorsingPeer’ and ‘chaincodeQuery’?

The former property is set to true when a particular peer has to act as an endorser, whereas the latter one is set to true when the peer has to be used for querying the state of the asset. Now, in order to understand this statement, you first need to understand the difference between ‘invoking a transaction’ and ‘querying the world state’, also known as a ‘query proposal’ in technical terms.

Querying simply means retrieving the data/world state from the ledger. For example, you can query the current state of a ‘student’ using a query proposal. A query proposal is invoked from the CLI command using the command ‘peer chaincode query’. You may refer to this [link](#) to learn more about this.

Having understood the meaning of ‘query proposal’, we will now understand the difference between ‘invoking a transaction’ to query the state of an asset, like we did for the certnet chaincode, and querying the world state using a ‘query proposal’.

When you query a chaincode using a ‘query proposal’, a transaction is not generated. If you recall from the previous module when you invoked the `getStudent()` function, the student details were returned to you and a transaction was generated. However, if you want to get the details without generating a transaction, then you need to query the chaincode, instead of invoking the function.

When the property `chaincodeQuery` is set to true for any peer, it means that the peer will be sent query proposals. The client application can use this peer to access the world state. You cannot set this property to true if that peer does not have the chaincode installed on it.

What happens when the property ‘ledgerQuery’ is set to true?

When this property is set to true for a particular peer, then that peer will be sent query proposals that do not require the chaincode. For example, `queryTransaction()` is used to query the blockchain to get the details of a particular transaction. This transaction does not require the chaincode to be installed on the peer.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Common Connection Profile – II:

- peers:

properties pertaining to each of the peers will be defined as below:

: url: this will be the url using with the client application will be able to connect to the peer using the grpc protocol

: grpcOptions: this will specify the override for the ssl target name and the timeout

: tlsCACerts: path of the TLS certificate

- certificateAuthorities:

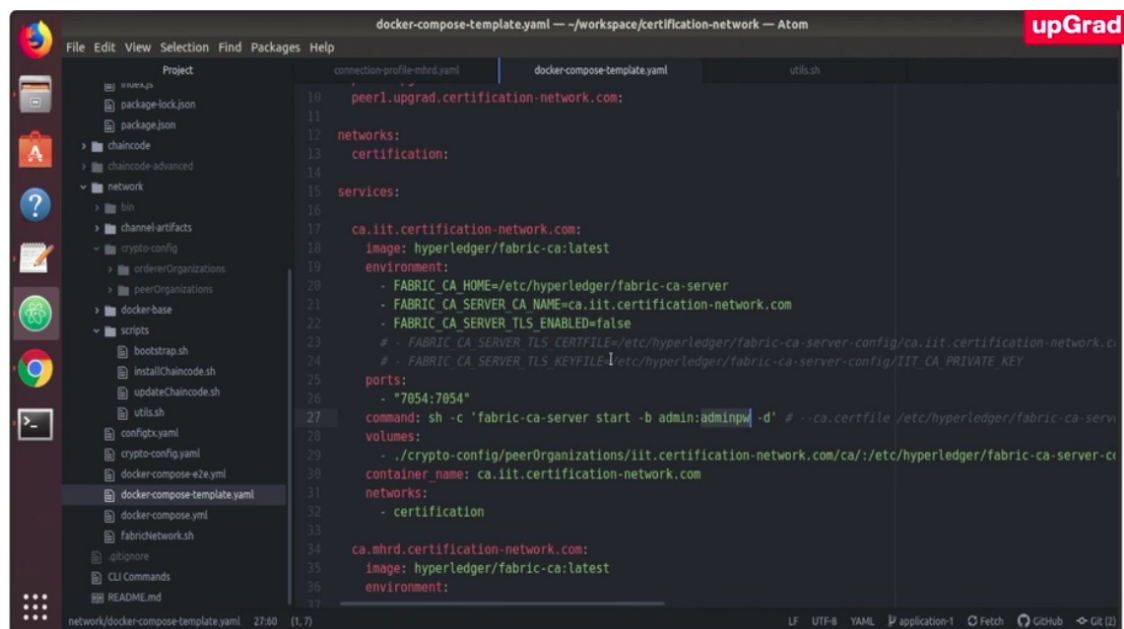
this defines the properties associated with the certificate authorities within the network

: url: this is the url of the CA of each organization. This will only use the *http* protocol and not *grpc*. Use *https* if the network is TLS enabled.

: httpOptions: The *verify* property checks whether to verify if the connection must comply to *https* if defined

: tlsCACerts: path for the CA certificates

: registrar: username and password for the client application to be able to login to this CA. The same username and password when the CA containers were started must be used here. This will have been set in the docker compose YAML configuration file in the *command* property



```
docker-compose-template.yaml -- ~/.workspace/certification-network — Atom
File Edit View Selection Find Packages Help
Project
  package-lock.json
  package.json
  chaincode
  chaincode-advanced
  network
  bin
  channel-artifacts
  crypto-config
  ordererOrganizations
  peerOrganizations
  docker-base
  scripts
    bootstrap.sh
    installChaincode.sh
    updateChaincode.sh
    utils.sh
    configtx.yaml
    crypto-config.yaml
    docker-compose-e2e.yaml
    docker-compose-template.yaml
    docker-compose.yaml
    fabricNetwork.sh
  .gitignore
  CLI Commands
  README.md
  network/docker-compose-template.yaml 27:60 (1, 7)
connection-profile-mhrd.yaml
docker-compose-template.yaml
utils.sh
10 peer1.upgrad.certification-network.com:
11
12 networks:
13   certification:
14
15 services:
16
17   ca.iit.certification-network.com:
18     image: hyperledger/fabric-ca:latest
19     environment:
20       - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
21       - FABRIC_CA_SERVER_CA_NAME=ca.iit.certification-network.com
22       - FABRIC_CA_SERVER_TLS_ENABLED=false
23       # - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.iit.certification-network.c
24       # - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-config/IIT_CA_PRIVATE_KEY
25     ports:
26       - "7054:7054"
27     command: sh -c 'fabric-ca-server start -b admin:adminpw -d' # --ca.certfile /etc/hyperledger/fabric-ca-serv
28     volumes:
29       - ./crypto-config/peerOrganizations/iit.certification-network.com/ca/:/etc/hyperledger/fabric-ca-server-c
30     container_name: ca.iit.certification-network.com
31     networks:
32       - certification
33
34   ca.mhrd.certification-network.com:
35     image: hyperledger/fabric-ca:latest
36     environment:
```

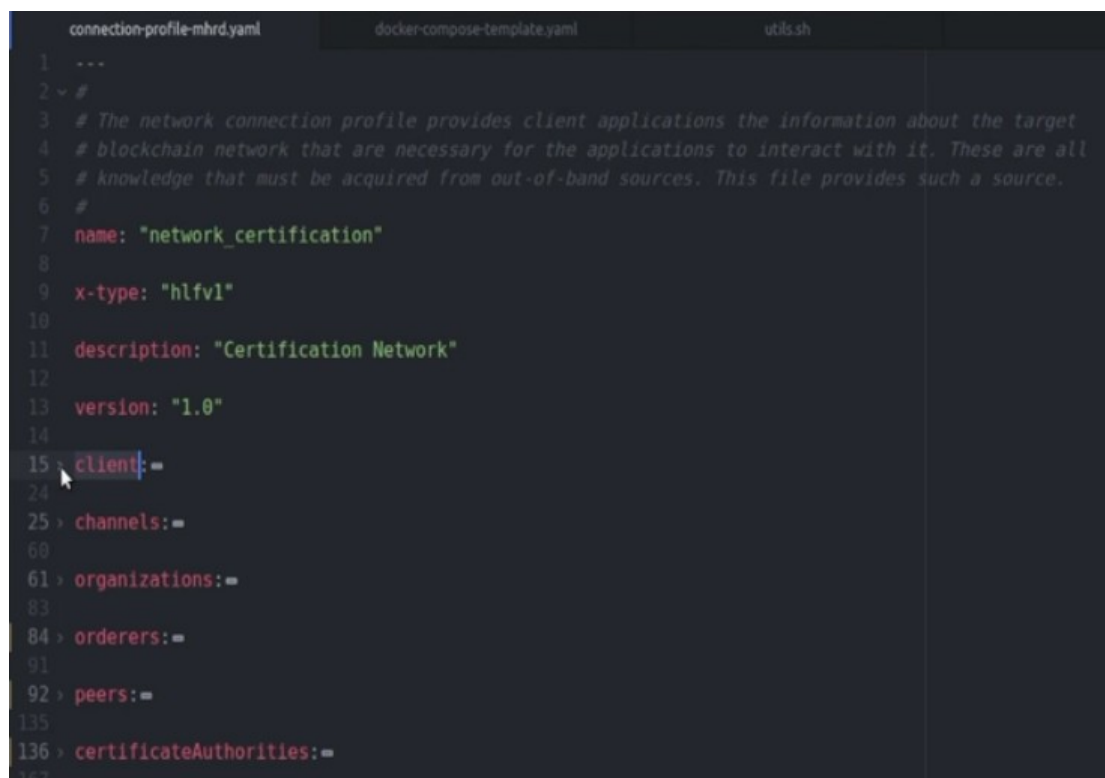
HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

- client:

this property specifies the organization to which this client application belongs. Each organization may have its own client application and the network needs to identify them accordingly. This property makes this CCP unique to the organization specified. This property is mandatory. This property captures the *organization* name and also the *connection timeout*.



```
connection-profile-mhrd.yaml  docker-compose-template.yaml  utils.sh
1  ---
2  #
3  # The network connection profile provides client applications the information about the target
4  # blockchain network that are necessary for the applications to interact with it. These are all
5  # knowledge that must be acquired from out-of-band sources. This file provides such a source.
6  #
7  name: "network_certification"
8
9  x-type: "hlfv1"
10
11 description: "Certification Network"
12
13 version: "1.0"
14
15 client: =
16
17 channels: =
18
19 organizations: =
20
21 orderers: =
22
23 peers: =
24
25 certificateAuthorities: =
```

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Node Applications

This section covers the different node modules and the way to run these modules using the terminal.

package.json:

Since this is a Node.js application, it is essential to have a package.json file which will define the dependencies for this project.

- name:

this will be the name for this application. Here we have used 'certification_network' in this case

- version:

this denotes the version for this application

- description:

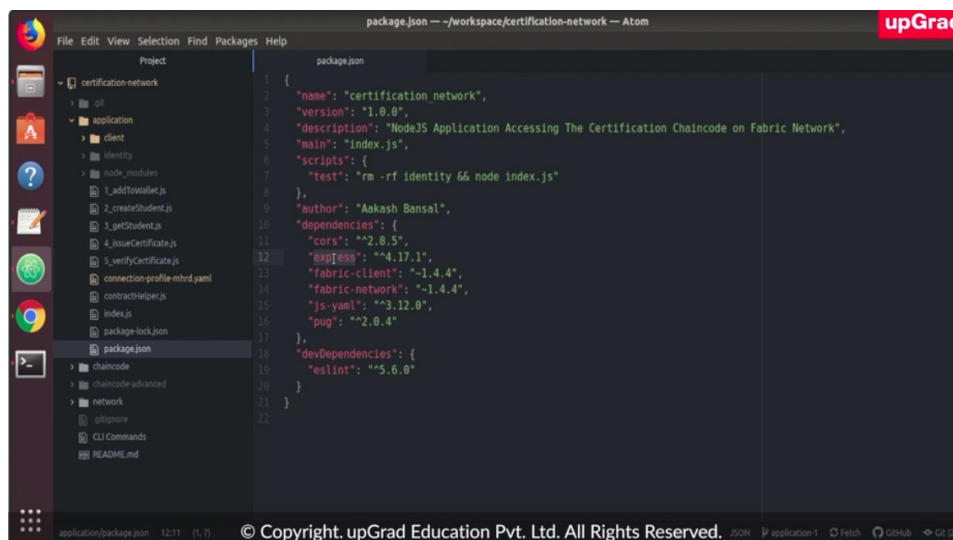
this is the description for this application

- main:

this is starting file for this project and in this case, we have specified this to be 'index.js'. Here all the other modules in this project will be exported and accordingly imported into this index.js file

- dependencies:

the most important dependencies will be *fabric-client* and *fabric-network*. These are the npm package files used in this project which gives the ability to interact with the network and access the ledgers on the peers. The other dependencies used will be 'js-yaml', 'express', 'pug' and 'cors' which will enable building a Node Express server



The screenshot shows the Atom editor interface with the 'package.json' file open. The left sidebar displays a file explorer with the project structure: 'certification-network' (root), 'application' (subdirectory), 'client' (subdirectory), 'identity' (subdirectory), 'node_modules' (subdirectory), and 'chaincode' (subdirectory). The 'package.json' file is selected in the sidebar. The main editor area shows the following JSON content:

```
1 {
2   "name": "certification network",
3   "version": "1.0.0",
4   "description": "NodeJS Application Accessing The Certification Chaincode on Fabric Network",
5   "main": "index.js",
6   "scripts": {
7     "test": "rm -rf identity && node index.js"
8   },
9   "author": "Aakash Bansal",
10  "dependencies": {
11    "cors": "^2.8.5",
12    "express": "^4.17.1",
13    "fabric-client": "~1.4.4",
14    "fabric-network": "~1.4.4",
15    "js-yaml": "^3.12.0",
16    "pug": "^2.0.4"
17  },
18  "devDependencies": {
19    "eslint": "^5.6.0"
20  }
21 }
```

The footer of the Atom editor shows the copyright notice: "© Copyright. upGrad Education Pvt. Ltd. All Rights Reserved."

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Creating Identity:

The first step is to create the wallet and store the identities required for the application to initiate transactions on the network. '1_addToWallet.js' has the code to create the credentials using the same certificates which were created as a part of the cryptogen command.

- setup the file system:
 - use the inbuilt 'fs' node module which provides functionalities to access the local file system
- create instances of the 'fabric-network' module:
 - create instances of the FileSystemWallet and X509WalletMixin classes that belong to the 'fabric-network' module. These will be used to create the certificates and add to the wallet
- define a directory for the wallet:
 - use the FileSystemWallet class to get an instance and create the wallet directory. The input parameter for this constructor will be the path of the directory
- main function:
 - this is the primary async function and could have any name
- get instance of certificate:
 - use the *readFileSync* method provided by the *fs* object to get one certificate from the crypto-config directory which will be used along with the private key to create the identity of the individual client. The file paths of the certificate and key will be inputs to the main function. You have to pick the correct certificate from the 'users' directory of the folder meant for that particular peer. The 'MSP' folder within the 'Admin' or any other user directory will have the 'signcerts' folder which has the certificate. The 'keystore' directory will have the private key
- define the label for the identity:
 - create the username for the identity stored in the wallet
- create the identity:
 - use the *createIdentity of the X509WalletMixin* method to create the identity. Three parameters are required for this method are the MSP ID, the certificate and the private key
- import the identity into the wallet:
 - use the *import* method of the *wallet* instance to import the identity into the wallet
- try, catch block:
 - include all the steps above in a standard try-catch block
- export the main function:
 - use the *module.exports* object to export the main function

HYPERLEDGER FABRIC & COMPOSER

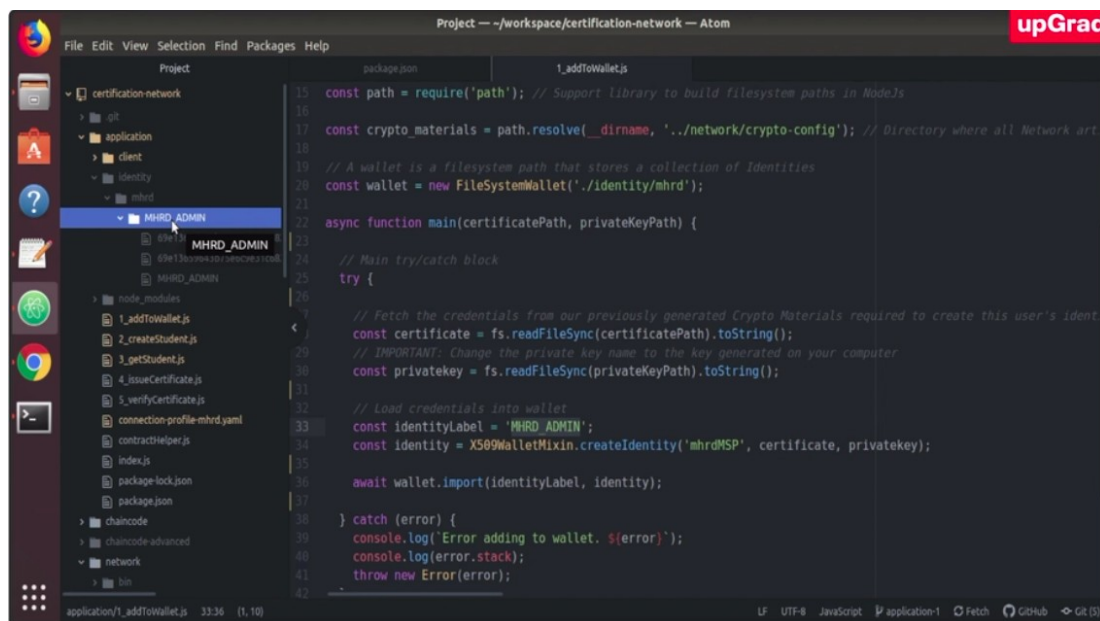
MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

To test out this module, call the *main* function with parameters of the certificate and the private key. Follow the steps in the documentation provided on the portal for the environment setup. Run this node module from the command line:

```
$ node 1_addToWallet.js
```

This will create a folder named *identity* within the *application* directory. This will have a folder named *MHRD_ADMIN* which is the label that was used to create this identity. There will be three files which will consists of the private key, public key and a certificate file.



Creating a student:

The node module '2_createStudent.js' has the logic to create a new student on the blockchain. The steps to do this are as below:

- get an instance of the gateway
- use this gateway to connect to the network
- get an instance of the channel
- get an instance of the chaincode/smart contract
- submit transactions

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Please refer to the code from the portal and following is the code walk-through:

- get instances of inbuilt modules:

- get instance of 'fs', 'js-yaml' and 'fabric-network' modules to enable the steps to connect to the Fabric network

- define the main function:

- this should be an async function since most of the methods used to connect with the network involves returning a promise. This function will take three parameters *studentId, name, email*

- get an instance of the contract:

- all the steps mentioned initially above have been written as a separate function named *getContractInstance()*. The steps are as below:

- : get an instance of the *Gateway* class

- : get an instance of the wallet where the identity has been stored (*./identity/mhrd*)

- : define a name for the identity

- : parse the YAML connection profile using the *yaml.safeLoad* method and store it in a variable

- : define connection options as an object which will have the properties such as *wallet, identity and discovery*

- : connect to the gateway using the *connect* method which takes in two parameters, *connection profile and connection options object* which have been defined before

- : use the *getNetwork* method to get an instance of the channel (*certificationchannel*)

- : use the *getContract* method provided by the channel instance to get an instance of the contract. The parameters to this will be the name of the chaincode (*certnet*) and the name of the smart contract (*org.certification-network.cernet*)

- submit transaction:

- use the *submitTransaction* method provided by the Fabric SDK. The parameters to this method are the *name of the function within the contract* and the *parameters of that function (studentId, name, email)*. These parameters are also being passed as an input the *main()* function. This step will fully execute the complete transaction flow on the blockchain network. This function returns a buffer stream

- convert the buffer stream to json:

- use the *JSON.parse* method to convert the buffer stream to json

- print this output to the console

- disconnect from the gateway using the *disconnect()* method

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

The main function call is also a part of the code which already has the input parameters which are the values to be recorded on the blockchain. Execute this module from the terminal as below:

\$ node 2_createStudent.js

```
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node 1_addToWallet.js
User identity added to wallet.
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node 2_createStudent.js
.....Connecting to Fabric Gateway
.....Connecting to channel - certificationchannel
.....Connecting to Certnet Smart Contract
.....Create a new Student account
.....Processing Create Student Transaction Response

{ studentId: '200',
  name: 'Aakash Bansal',
  email: 'connect@aakashbansal.com',
  school: 'xs09:/C=US/ST=California/L=San Francisco/OU=admin/CN=Admin@nhrd.certification-network.com:/C=US/ST=California/L=San Francisco/O=nhrd.certification-network.com/CN=ca.nhrd.certification-network.com',
  createdAt: '2019-10-09T15:20:26.005Z',
  updatedAt: '2019-10-09T15:20:26.005Z' }

.....Create Student Transaction Complete!
.....Disconnecting from Fabric Gateway
Student account created
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$
```

Getting Student Details:

Since the steps in the `getContractInstance()` function will need to be repeated in all the functions, this function will be moved to a separate node module named *contracthelper.js*. This will enable reducing the redundancy in the code. Hence this module will only have the `main()` function.

- get instance of the contract helper:
 - import the contract helper using the `require` keyword
- define the main function
 - the main function will take in the `studentId` as the input parameter
- get instance of the contract:
 - using the contract helper class which has been defined separately, obtain the instance of the contract and store it in a variable named *certnetContract*
- submit the transaction
 - use the *submitTransaction* function to invoke the *getStudent* function. Here, the `studentId` is passed as an input parameter. The result will be a buffer stream which will be stored in a variable named *studentBuffer*
- convert the buffer to string
 - use `JSON.parse` and `toString()` method to convert the data buffer into string
- print and return this value as output

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Call the main function with value as '200' as the studentID. From the terminal, run this module using the following command:

\$ node 3_getStudent.js



```
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node 3_getStudent.js
.....Connecting to Fabric Gateway
.....Connecting to channel - certificationchannel
.....Connecting to Certnet Smart Contract
.....Get Student Account
.....Processing Get Student Transaction Response

{ studentId: '200',
  name: 'Aakash Bansal',
  email: 'connect@akashbansal.com',
  school: 'x509:/C=US/ST=California/L=San Francisco/OU=admin/CN=Admin@mhrd.certification-network.com:/C=US/ST=California/L=San Francisco/O=mhrd.certification-network.com/CN=ca.mhrd.certification-network.com',
  createdAt: '2019-10-09T15:20:26.005Z',
  updatedAt: '2019-10-09T15:20:26.005Z' }

.....Get Student Transaction Complete!
.....Disconnecting from Fabric Gateway
.....API Execution Complete!
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$
```

Issue Certificate:

This module creates a certificate asset on the network and following are the steps:

- get instance of the helper:
 - get instance of the helper function
- define main() function
 - define the main function which will be an async function with input parameters as *studentId, courseID, grade, hash*
- get instance of the contract:
 - get instance of the contract using the helper function
- submit transaction
 - use the *submitTransaction* method to call the *issueCertificate* function and pass in the parameters *studentID, courseID, grade and hash*. This returns a data buffer stored in a variable
- convert the buffer to string:
 - use *JSON.parse* and *toString()* method to convert the data buffer to string
- print and return this data to the user

Call the main() function by passing values of all the parameters. Here we use, '200', 'PGCBC', 'A', 'asdfgh'. From the terminal, run this module as below:

\$ node 4_issueCertificate.js

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

```
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node 4_issueCertificate.js
.....Connecting to Fabric Gateway
.....Connecting to channel - certificationchannel
.....Connecting to Certnet Smart Contract
.....Issue Certificate To Student
.....Processing Issue Certificate Transaction Response

{ studentId: '200',
  courseId: 'PGDBC',
  teacher: 'x509::/C=US/ST=California/L=San Francisco/OU=admin/CN=Admin@mhrd.certification-network.com:/C=US/ST=California/L=San Francisco/O=mhrd.certification-network.com/CN=ca.mhrd.certification-network.com',
  certId: 'PGDBC-200',
  originalHash: 'asdfgh',
  grade: 'A',
  createdAt: '2019-10-09T15:38:15.990Z',
  updatedAt: '2019-10-09T15:38:15.990Z' }

.....Issue Certificate Transaction Complete!
.....Disconnecting from Fabric Gateway
Certificate created for the student
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$
```

Verifying the certificate:

This module will be used to verify the certificate of the student using the following steps:

- get instance of the helper:
 - get instance of the helper module
- define the main() function:
 - this will again be async function which has *studentID*, *courseID*, *hash* as its input parameters
- get the instance of the smart contract:
 - get an instance of the smart contract using the helper function
- submit transaction
 - using the *submitTransaction* method, invoke the *verifyCertificate* function by passing the parameters *studentID*, *courseID*, *hash*
- pass the response of the event to the console:
 - since this function within the smart contract triggers an event, use the *addContractListener* method to subscribe to the event and obtain the data. The input parameters to this method is the name of the event and the function name. The callback function will obtain the payload of the event by using the *event.payload* method. This payload will be a buffer stream
- covert the buffer stream as string:
 - use *JSON.parse* and *toString()* methods to convert the data to string and print this on the console
- Call the main function and pass the parameters such as *studentID*, *courseID* and *hash*. For example, here it could be '200', 'PGDBC', 'asdfgh'.

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Run this module from the terminal as below. We will see that the result will be printed as VALID for the set of student and certificate data provided as inputs to this module.

\$ node 5_verifyCertificate.js

```
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node 5_verifyCertificate.js
.....Connecting to Fabric Gateway
.....Connecting to channel - certificationchannel
.....Connecting to Certnet Smart Contract
.....Verify Certificate Of Student

*** NEW EVENT ***
{ chaincode_id: 'certnet',
  tx_id: '09cccc79f777069ad8c579528340e8faef7f3cb593109b0c780775a23b675597',
  event_name: 'verifyCertificate',
  payload:
    { certificate: 'PGDBC-200',
      student: '200',
      verifier: 'x509::/C=US/ST=California/L=San Francisco/OU=admin/CN=Admin@mhrd.certification-network.com::/C=US/ST=California/L=San Francisco/OU=mhrd.certification-network.com/CN=ca.mhrd.certification-network.com',
      result: '*** - VALID',
      verifiedOn: '2019-10-09T15:45:34.668Z' } }

.....Disconnecting from Fabric Gateway
Verification result available
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$
```

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Node Server & UI

This module covers creating a node server and a front-end interface for the application.

So far, the application was being run from the terminal which is how the end user would have to use these functions. However, to make it more user friendly, we may have to build an interface using a web browser rather than running CLI commands.

Applications on web server will be built in the form of API's which are specific end points or URL's. So, any function which have been defined as independent node modules will now be converted as API's which can be accessed by a client website using HTTP requests. To enable this, a HTTP server must be created. We will use a node.js based HTTP server. This will be available on the local host at port 3000. We will use a node.js library called Express which is a framework to build the HTTP server. Each node module will be linked to a URL and when triggered will call and execute that specific function.

Server creation:

The code for creating the server will be defined in the *index.js* file which is the main starting point for this application. Following are the steps and the code walk-through.

- get an instance of the Express module:
this enables to quickly create a node.js server
- get an instance of the cors module:
this helps to overcome restrictions of the Chrome browser and enable access files on the local computer
- define the port number as 3000
- import all the node modules and store them in variables
- use the *cors* policy by using the *use* method to add properties to the server:
 - : enable conversion to json
 - : to enable parsing url encoded data
- set the title for the application
- define the end points for the server:
 - : use the instance of the express (*app*) and the HTTP methods like GET, POST etc by specifying the path

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

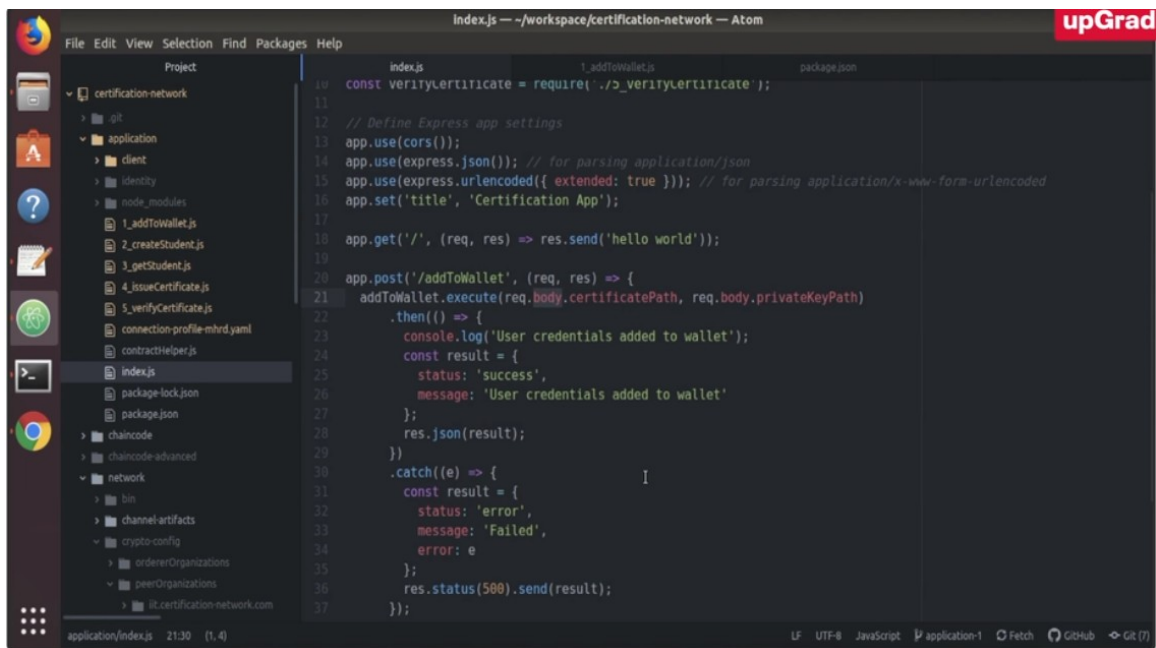
: using the variables of the imported modules, call the *execute* function which have been exported in each of the modules. The input parameters for this will be the *certificate path* and the *private key path*. The input parameters in the body of the HTTP requests will be used by the server to run the logic and get a response back

: the result is available as json

- repeat the same for each of the API's or functions

- start the express server:

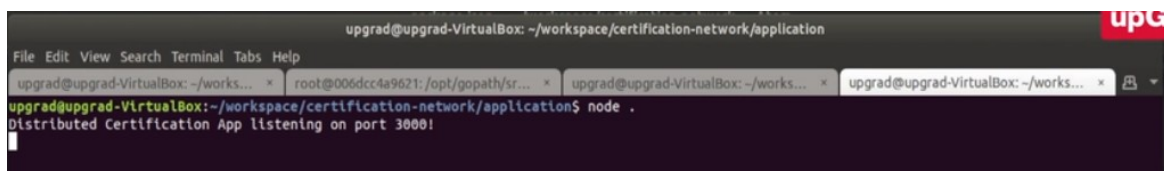
use the *listen* method to start the node server



```
index.js
10 const verifyCertificate = require('./_verifyCertificate');
11
12 // Define Express app settings
13 app.use(cors());
14 app.use(express.json()); // for parsing application/json
15 app.use(express.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded
16 app.set('title', 'Certification App');
17
18 app.get('/', (req, res) => res.send('hello world'));
19
20 app.post('/addToWallet', (req, res) => {
21   addToWallet.execute(req.body.certificatePath, req.body.privateKeyPath)
22     .then(() => {
23       console.log('User credentials added to wallet');
24       const result = {
25         status: 'success',
26         message: 'User credentials added to wallet'
27       };
28       res.json(result);
29     })
30     .catch(e => {
31       const result = {
32         status: 'error',
33         message: 'Failed',
34         error: e
35       };
36       res.status(500).send(result);
37     });
38 })
```

- start the server by running the default node application from the terminal and access the HTTP service from the web browser

\$ node .



```
upgrad@upgrad-VirtualBox: ~/workspace/certification-network/application
upgrad@upgrad-VirtualBox:~/workspace/certification-network/application$ node .
Distributed Certification App listening on port 3000!
```



HYPERLEDGER FABRIC & COMPOSER

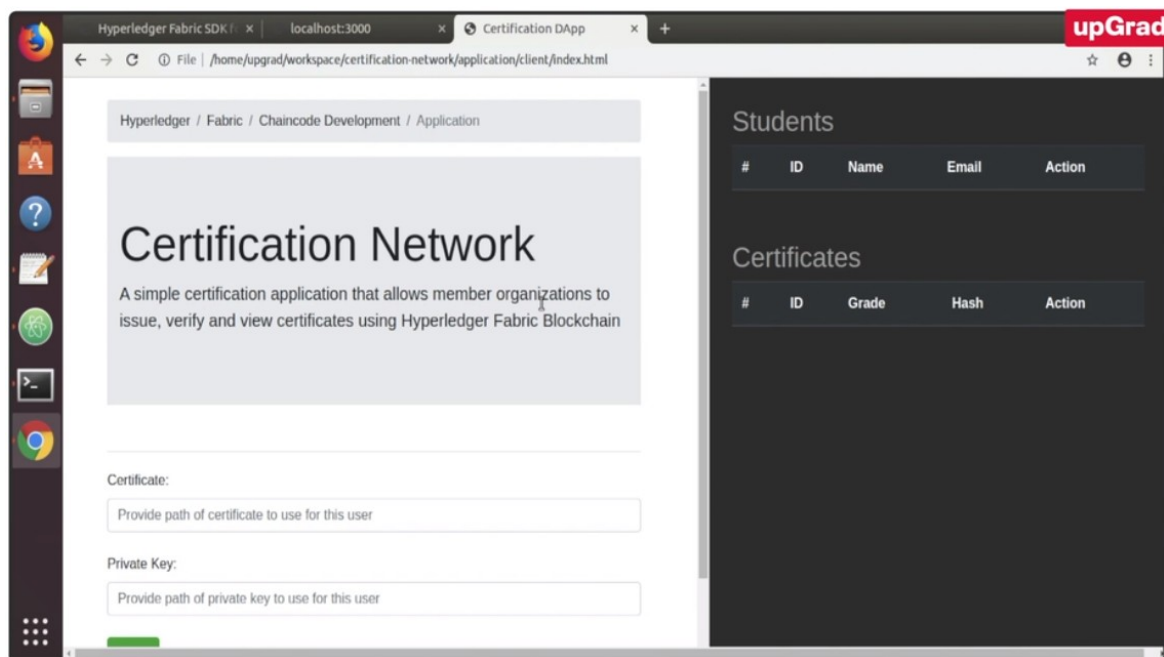
MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Building the user interface I:

In terms of the data flow, the user interface will trigger HTTP requests to the API's which are accessible through the node server. The API's will run the node module which in turn will access the Fabric network and invoke a transaction using the smart contract function. This response value is sent back to the API and displayed on the user interface.

The *client* folder has all the html, css and javascript files to build a web user interface. The layout would look as below when opened from a web browser. The layout has functionalities to login and display the student and certificate details.



- This page uses the *getElementById* method to obtain the certificate and private key paths from the user interface
- use the jquery library to post the HTTP request and hit the URL as appropriate to the function/module being invoked
- when the login button is hit, a new user identity is created in the wallet
- once the response is obtained, make changes to the user interface to let the user know that the login has been successful

HYPERLEDGER FABRIC & COMPOSER

MODULE 4

APPLICATION DEVELOPMENT ON HYPERLEDGER FABRIC

Building the user interface II:

Similar to the login interface, the create student interface captures the required input parameters. It uses the *geElementById* method to capture the values from the form and use the jquery method to post the HTTP request to the end points named *newStudent* and *issueCertificate* respectively.

The result of this HTTP call is processed within the API to access the smart contract function. Once the HTTP response is obtained, the user interface is updated with the new student and certificate details. The *verify* button enables certificate verification.

