

< client 웹페이지 만들기 >

2018. 11. 22(목)

1. 네트워크 구동하기(사전작업)

- client(SDK)를 구성하는건 go, nodejs, javascript 3개 언어로 가능!
- 인증서가 있어야 페브릭에 접속 가능
- \$./generate.sh를 하면 인증서 키가 생성됨(이게 있어야 user 등록이 가능함)

```
guru@guru:~/tuna-github/education/LFS171x/fabric-material/tuna-app$ ll ~/.hfc-key-store/
total 20
drwxrwxr-x 2 guru guru 4096 Nov 22 14:04 ./
drwxr-xr-x 13 guru guru 4096 Nov 21 17:53 ../
-rw-rw-r-- 1 guru guru 986 Nov 22 14:04 admin
-rw-rw-r-- 1 guru guru 246 Nov 22 14:04 cf5d4e190f1a4ce1d922acafea9995751eff64042cb2d1aa3b99fdfe40ee936b-priv
-rw-rw-r-- 1 guru guru 182 Nov 22 14:04 cf5d4e190f1a4ce1d922acafea9995751eff64042cb2d1aa3b99fdfe40ee936b-pub
```

- startFabric.sh가 안되면? 체인코드 오류
registerUser.js가 안되면? 인증서 오류(generate)
- npm install은 package가 있는 곳에서 해야 함!
- /tuna/basic-network에 있는 docker-compose.yml은 네트워크 설정해주는 기본파일

```
#
# Copyright IBM Corp All Rights Reserved
#
# SPDX-License-Identifier: Apache-2.0
#
version: '2'

networks:
  basic:

services:
  ca.example.com: ...

  orderer.example.com: ...

  peer0.org1.example.com: ...

  couchdb: ...

  cli: ...
```

order, peer, db, cli관련 설정이 코딩되어 있음

: peer 안에 환경변수에 작성된 KEYFILE의 뒷부분(해쉬값)과 \$ tree crypto-config해서 뜨는 정보 중 peer 아래 있는 ca값과 일치해야 인증이 가능!!(이게 다르면 user 등록 시 오류 발생) 복붙과정에서 바뀔 수 있기 때문에 다른 경우 아래 값을 복사해서 위에 붙여 넣고 저장

```
10
11 services:
12   ca.example.com:
13     image: hyperledger/fabric-ca
14     environment:
15       - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
16       - FABRIC_CA_SERVER_CA_NAME=ca.example.com
17       - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem
18       - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/4239aa0dcd76daeeb8ba0cda701851d14504d31aad1b2dddbac6a57365e497c_sk
19     ports:
20       - "7054:7054"
21     command: sh -c 'fabric-ca-server start -b admin:adminpw'
22     volumes:
23       - ./crypto-config/peerOrganizations/org1.example.com/ca:/etc/hyperledger/fabric-ca-server-config

peerOrganizations
├── org1.example.com
│   ├── ca
│   │   ├── 93c35ac5376eb1c2b6a2dbc24f282bd8bad95ebf853b29e60e026cae09e9df68_sk
│   │   └── ca.org1.example.com-cert.pem
│   └── msp
│       ├── admincerts
│       │   └── Admin@org1.example.com-cert.pem
│       ├── cacerts
│       │   └── ca.org1.example.com-cert.pem
│       ├── tlscacerts
│       │   └── tlscacert.org1.example.com-cert.pem
```

- ```
guru@guru:~/tuna/tuna-app$ node registerAdmin.js
Store path:/home/guru/.hfc-key-store
(node:6771) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully enrolled admin user "admin"
Assigned the admin user to the fabric client ::{"name":"admin","mspid":"Org1MSP","roles":null,"affiliation":"","enrollmentSecret":"","enrollment":{"signingIdentity":"5f2dbf1f310ae69fb1ce7fd66d1827d8247c2b24e1714ecd9ff42a26cc57b0dd"},"identity":{"certificate":"-----BEGIN CERTIFICATE-----\nMIICATCCAaigAwIBAgIUOpich+SvH4iIjbszUKNi0eRdBCowCgYIKoZIzj0EAeIwIwczELMAkGAUUEBhQCMVwEzARBgNVBAGTCkNhbmB1bmB3JuaWExfjAUBGNVBACDVBh\n/nb1BGcmFuY2VyY2Z8Y28xGTABG8NVBAOTEg9yZzEuZDZ8XhhbXBSZS5jb20xHDAaBgNVBAMT\n/nE2NlM9yZzEuZDZ8XhhbXBSZS5jb20xHhNMTGxMTiYMDU0ZOTAwHhcnMTkxMTiYMDU0\n/nDAwMjA1hQJ08wQDQVYDQLEWZjb1lnbG9xXjAMBGBNVBAMTBWFKbWUuMfkwEYHk0ZI\n/nzj0CAQYIKoZIzj0DAQDQAgEYU1rkkCNQPT0ioi0FTBZjwT1sK2V3wsrd4Fudfw\n/nBeZqBViIr/DZ6geAX3yFXdM1p0jEq+6v6CX00rYugmqvqNsMgowDgYDVR0PAQH/\n/nBAQDAgeAAWAwA1UdEWEb/wQCMAAwHQYDVR0RB0BEEFADTCysJpeDgAjcdUaht8Kc\n/n3d7nMCSGA1UdIwQ1MCKCAIYDVsU3brHcTqLbwk80r3i6Zv6/tSp5g4CbK4J6d9o\n/nMAoGCCqGSM49BAMCA0CAMEQCI8A0eh4pgyKHnLS1SDGKM8abgp15dL+HynK8zJwT\n/nWE1AiBAYDvbsErvEABA+ZfzjgCoVYHpv+XqSpkQob0VZL3g=\n\n-----END CERTIFICATE-----\n"}}}
```

- ```
guru@guru:~/tuna/tuna-app$ node registerUser.js
Store path:/home/guru/.hfc-key-store
(node:6789) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Successfully loaded admin from persistence
Successfully registered user1 - secret:eprrhbSaQxfw
Successfully enrolled member user "user1"
User1 was successfully registered and enrolled and is ready to intreact with the fabric network
```

- ```
Channels peers has joined:
mychannel
Get instantiated chaincodes on channel mychannel:
Name: tuna-app, Version: 1.0, Path: github.com/tuna-app, Escc: escc, Vscc: vscc
```

- ```
guru@guru:~/tuna/tuna-app$ ll
total 292
drwxrwxr-x   3 guru guru  4096 Nov 22 11:58 ./
drwxrwxr-x   5 guru guru  4096 Nov 21 17:12 ../
-rw-rw-r--   1 guru guru 20935 Nov 21 17:37 controller.js
drwxrwxr-x 346 guru guru 12288 Nov 21 17:54 node_modules/
-rw-rw-r--   1 guru guru   579 Nov 22 11:58 package.json
-rw-rw-r--   1 guru guru 228240 Nov 21 17:54 package-lock.json
-rw-rw-r--   1 guru guru  2939 Nov 21 17:37 registerAdmin.js
-rw-rw-r--   1 guru guru  3261 Nov 21 17:37 registerUser.js
-rw-rw-r--   1 guru guru   462 Nov 21 17:37 routes.js
-rw-rw-r--   1 guru guru  1204 Nov 21 17:37 server.js
-rwxrwxr-x   1 guru guru  1820 Nov 21 17:37 startFabric.sh*
```

: package.json은 nodejs에 필요한 걸 모아놓은 놈 \$ **npm install**로 필요한 것들 자동설치하면 node modules라는 디렉토리가 생성됨

: client에 웹서버 관련 코드가 다 있음(angular로 작성됨 → 우리는 장고로 작성해서 올리면 됨)

2. 웹서버 구동하기

- tuna/tuna-app 안에 있는 파일들의 역할 설명

① **server.js** : 장고의 runserver 역할을 하며 서버 구동시킴(nodejs로 작성됨 / express)

=> \$ node server.js

```
29 // set up a static file server that points to the "client" directory
30 // app.use(express.static(path.join(__dirname, './client')));
31
32 // Save our port
33 var port = process.env.PORT || 8000;
34
35 // Start the server and listen on port
36 app.listen(port, function(){
37   console.log("Live on port: " + port);
38 });
```

: 30번째 줄 주석처리 해줄 것(앵귤러 이용하는건데 우린 이거 안 쓸꺼기 때문에 우선 막기)

: app.listen이 runserver 역할 해주는 함수

: cmd창에서 실행하면 아래 사진처럼 작동(8000포트로 연결되어 계속 listen 중)

```
guru@guru:~/tuna/tuna-app$ node server.js
(node:6942) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Live on port: 8000
```

: 이 cmd창을 끄면 안됨(서버 끊기므로) => 다른 명령어 쓰려면 새로운 창 열어서 실시!

② **controller.js** : 장고의 views 역할을 하며 서버 구동시킴(chaincode와 연결해주는 파일)

=> \$ node controller.js

```
// queryAllTuna - requires no arguments , ex: args: [''],
const request = {
  chaincodeId: 'tuna-app',
  txId: tx_id,
  fcn: 'queryAllTuna',
  args: ['']
};
```

```
module.exports = (function() {
  return{
    get_all_tuna: function(req, res){
      console.log("getting all tuna from database: ");

      var fabric_client = new Fabric_Client();

      // setup the fabric network
      var channel = fabric_client.newChannel('mychannel');
      var peer = fabric_client.newPeer('grpc://localhost:7051');
      channel.addPeer(peer);

      //
      var member_user = null;
      var store_path = path.join(os.homedir(), '.hfc-key-store');
      console.log('Store path:' + store_path);
      var tx_id = null;
```

: chaincode에서 작성한 함수들과 연결(success, error 메시지 작성 등)

: 연결해주는 역할을 하는 것이 grpc(nodejs sdk를 하기 위해 \$ npm install)

: 각 함수별 시작부분에 localhost와 채널이름 작성 / .hfc 파일에 인증서 들어감

: request에는 실제 필요 정보가 입력됨

③ **routes.js** : 장고의 urls.py 역할(PATH 정보 알려주는 파일)

=> \$ node routes.js

```
//SPDX-License-Identifier: Apache-2.0
var tuna = require('./controller.js');

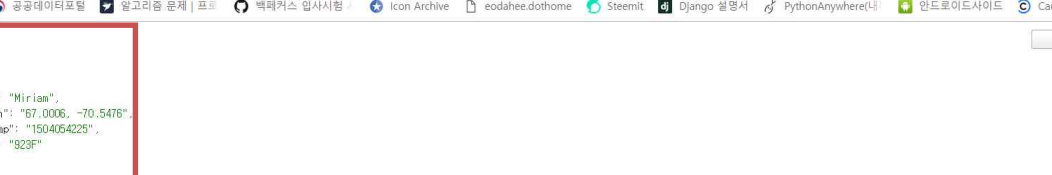
module.exports = function(app){

  app.get('/get_tuna/:id', function(req, res){
    tuna.get_tuna(req, res);
  });
  app.get('/add_tuna/:tuna', function(req, res){
    tuna.add_tuna(req, res);
  });
  app.get('/get_all_tuna', function(req, res){
    tuna.get_all_tuna(req, res);
  });
  app.get('/change_holder/:holder', function(req, res){
    tuna.change_holder(req, res);
  });
}
```

: http://localhost:8000/get_all_tuna라고 치면 참치 정보 다 가져옴

: server.js(express)를 통해 node sdk에 전달되고 fabric을 호출해 chaincode 구동하는 것

- 웹서버 커서 url입력하면 작성한 함수에 해당하는게 보여짐! 너무신기ㅠㅠ



The screenshot shows a web browser with the address bar set to `localhost:8000/get_all_tuna`. The response is a JSON array of four records, each with a 'Key', a 'Record' object, and a 'vessel' string. The 'vessel' field is highlighted in red in the original image.

```
[{"Key": "1", "Record": {"holder": "Miriam", "location": "67.0006, -70.5476", "timestamp": "1504054225", "vessel": "823F"}, "vessel": "823F"}, {"Key": "10", "Record": {"holder": "Fatima", "location": "51.9435, 8.2795", "timestamp": "1487745091", "vessel": "49W4"}, "vessel": "49W4"}, {"Key": "2", "Record": {"holder": "Dave", "location": "91.2395, -49.4594", "timestamp": "1504057825", "vessel": "M83T"}, "vessel": "M83T"}, {"Key": "3", "Record": {"holder": "Igor", "location": "58.0148, 59.01991", "timestamp": "1493517025", "vessel": "1012"}, "vessel": "1012"}, {"Key": "4", "Record": {"holder": "Igor", "location": "58.0148, 59.01991", "timestamp": "1493517025", "vessel": "1012"}, "vessel": "1012"}]
```

- localhost:8000/get_all_tuna
- localhost:8000/get_tuna/1 : 뒤에 id 입력해주기
- localhost:8000/add_tuna/1-50,30-165879-dahee-kim : 뒤에 입력데이터 5개 값("-"로 구분)
key-location-timestamp-holder-vessel
- localhost:8000/change_holder/5-dahee : 뒤에 id, holder 입력(해당 id의 holder로 바꿈)

The screenshot shows the MongoDB Compass web interface. The browser address bar indicates the URL is `localhost:5984/_utils/#database/mychannel_tuna-app/_all_docs`. The interface is divided into a sidebar on the left and a main content area. The sidebar contains navigation links: 'All Documents', 'Run A Query with Mango', 'Permissions', 'Changes', and 'Design Documents'. The main content area shows a table of documents for the collection 'mychannel_tuna-app'. The table has columns: '_id', 'holder', 'location', 'timestamp', and 'vessel'. There are 9 documents listed. At the bottom of the interface, it says 'Showing 5 of 7 columns' and 'Showing document 1 - 10'.

_id	holder	location	timestamp	vessel
1	Miriam	67.0006, -70.5476	1504054225	923F
10	Fatima	51.9435, 8.2735	1487745091	49W4
2	Dave	91.2395, -49.4594	1504057825	M83T
3	Igor	58.0148, 59.01391	1493517025	T012
4	Amalea	-45.0945, 0.7949	1496105425	P490
5	Rafa	-107.6043, 19.5003	1493512301	S439
6	Shen	-155.2304, -15.8723	1494117101	J205
7	Lella	103.8842, 22.1277	1496104301	S22L
8	Yuan	-132.3207, -34.0983	1485066691	EI89
9	Carlo	153.0054, 12.6429	1485153091	129R

- VScode에서는 server창에(cmd) 호출한 함수가 성공적으로 호출되었다고 뜬!
(controller.js에서 지정한대로 뜬)

```
guru@guru:~/tuna/tuna-app$ node server.js
(node:6942) DeprecationWarning: grpc.load: Use the @grpc/proto-loader module with grpc.loadPackageDefinition instead
Live on port: 8000

getting all tuna from database:
Store path:/home/guru/.hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is [{"Key":"1", "Record":{"holder":"Miriam", "location":"67.0006, -70.5476", "timestamp":"1504054225", "vessel":"923F"}}, {"Key":"10", "Record":{"holder":"Fatima", "location":"51.9435, 8.2735", "timestamp":"1487745091", "vessel":"49W4"}}, {"Key":"2", "Record":{"holder":"Dave", "location":"91.2395, -49.4594", "timestamp":"1504057825", "vessel":"M83T"}}, {"Key":"3", "Record":{"holder":"Igor", "location":"58.0148, 59.01391", "timestamp":"1493517025", "vessel":"T012"}}, {"Key":"4", "Record":{"holder":"Amalea", "location":"-45.0945, 0.7949", "timestamp":"1496105425", "vessel":"P490"}}, {"Key":"5", "Record":{"holder":"Rafa", "location":"-107.6043, 19.5003", "timestamp":"1493512301", "vessel":"S439"}}, {"Key":"6", "Record":{"holder":"Shen", "location":"-155.2304, -15.8723", "timestamp":"1494117101", "vessel":"J205"}}, {"Key":"7", "Record":{"holder":"Leila", "location":"103.8842, 22.1277", "timestamp":"1496104301", "vessel":"S22L"}}, {"Key":"8", "Record":{"holder":"Yuan", "location":"-132.3207, -34.0983", "timestamp":"1485066691", "vessel":"EI89"}}, {"Key":"9", "Record":{"holder":"Carlo", "location":"153.0054, 12.6429", "timestamp":"1485153091", "vessel":"129R"}}]
```

4. 기타

- 장고와 이걸 연결하기 위해서는 JSON-Parsing을 알아야 함!!
JSON으로 받아오고 뿌려줘야하기 때문에 이걸 알아야 다른데서 연결이 가능!
- 블록이 생성되는 타임은 tuna/basic-network/configtx.yaml파일에 설정되어 있음(2초)

```
guru@guru:~/tuna/basic-network$ ls
config configtx.yaml crypto-config crypto-config.yaml docker-compose.yml generate.sh init.sh README.md start.sh stop.sh teardown.sh

Orderer: &OrdererDefaults

# Orderer Type: The orderer implementation to start
# Available types are "solo" and "kafka"
OrdererType: solo

Addresses:
- orderer.example.com:7050

# Batch Timeout: The amount of time to wait before creating a batch
BatchTimeout: 2s
```

- tuna-app 밑에 src 디렉토리에 보면 구현한 chaincode의 js파일이 들어가 있음
이 4가지 파일을 합친 것이 controller.js 파일!

```
guru@guru:~/tuna/tuna-app/src$ ll
total 32
drwxrwxr-x 2 guru guru 4096 Nov 22 16:50 ./
drwxrwxr-x 5 guru guru 4096 Nov 22 16:50 ../
-rw-rw-r-- 1 guru guru 7311 Nov 22 16:50 changeTunaHolder.js
-rw-rw-r-- 1 guru guru 2873 Nov 22 16:50 queryAllTuna.js
-rw-rw-r-- 1 guru guru 2865 Nov 22 16:50 queryTuna.js
-rw-rw-r-- 1 guru guru 7523 Nov 22 16:50 recordTuna.js
```

- queryAllTuna 함수 수정!(start, end값 받아서 해당 범위 내의 TUNA만 출력하도록)
chaincode 수정 후 docker에서 install → upgrade
routes.js 파일에 url 수정(입력값 받아오도록)
controller.js 파일에 인자 추가, request 수정
다 완료 후 invoke로 실시해보면 변경사항 적용 가능!!