< Groot_scan 페이지 구현 >

● 아이디어

- GetStateByRange를 이용해 최종 블록의 상태(값)를 가져옴(CouchDB에 있는 값)
  startkey와 endkey를 비워두면 전체 값 가져옴(지정도 가능)
- GetHistoryForKey를 이용해 각 Key별 history 조회 가능

▶ get_all_tech로 모든 블록을 받아와 key값 출력 후 각각의 key 값들을 get_cert_verify에 넣어 모든 history 출력
timestamp를 기준 내림차순으로 정렬(가장 최근 tx가 위로 올라오도록)
groot_scan페이지의 Block, Transaction에 각각 for문, parsing 사용 데이터 입력!

● Transaction 관련

1. 해당 channel에 쌓인 블록의 height를 알 수 있는 함수?
   - $ peer channel getinfo -c mychannel

```
root@dea15188aae5:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer channel getinfo -c mychannel
2019-01-29 01:29:40.522 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-01-29 01:29:40.582 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-01-29 01:29:40.588 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
Blockchain info: {"height":216,"currentBlockHash":"rcNpKZnAbPzZjAIzPuWlBEJl34trLdprZGf8fzKBKBg=","previousBlockHash":"K7TvRo/+6XLchnp23iHSWEizp6ospgTuymVowjQDKL8="}
```

   전체 블록의 길이와 이전블록, 현재 블록의 hash값 알 수 있음!!

2. $ docker logs peer0.org1.example.com 2>&1 | grep -i -a -E 'private|pvt|privdata'
   $ docker logs peer0.org1.example.com 2>&1 | grep -i 'Committed block'

```
groot@GROOT-Server4:~/groot/chaincode/groot_1.2$ docker logs peer0.org1.example.com 2>&1 | grep -i 'Committed block'
2018-12-28 01:31:02.674 UTC [kvledger] CommitWithPvtData -> INFO 02b [mychannel] Committed block [0] with 1 transaction(s) in 341ms (state_validation=1ms block_commit=124ms state_commit=185ms)
2018-12-28 01:31:23.912 UTC [kvledger] CommitWithPvtData -> INFO 045 [mychannel] Committed block [1] with 1 transaction(s) in 177ms (state_validation=12ms block_commit=119ms state_commit=18ms)
2018-12-28 01:31:34.279 UTC [kvledger] CommitWithPvtData -> INFO 04b [mychannel] Committed block [2] with 1 transaction(s) in 153ms (state_validation=4ms block_commit=118ms state_commit=6ms)
2018-12-28 02:07:43.832 UTC [kvledger] CommitWithPvtData -> INFO 067 [mychannel] Committed block [3] with 1 transaction(s) in 386ms (state_validation=57ms block_commit=151ms state_commit=122ms)
2018-12-28 04:30:09.830 UTC [kvledger] CommitWithPvtData -> INFO 0de [mychannel] Committed block [4] with 1 transaction(s) in 363ms (state_validation=89ms block_commit=125ms state_commit=107ms)
2018-12-28 04:32:19.530 UTC [kvledger] CommitWithPvtData -> INFO 0f5 [mychannel] Committed block [5] with 1 transaction(s) in 181ms (state_validation=18ms block_commit=122ms state_commit=19ms)
2018-12-28 04:40:17.151 UTC [kvledger] CommitWithPvtData -> INFO 10e [mychannel] Committed block [6] with 1 transaction(s) in 223ms (state_validation=39ms block_commit=121ms state_commit=35ms)
2018-12-28 04:50:22.756 UTC [kvledger] CommitWithPvtData -> INFO 11f [mychannel] Committed block [7] with 1 transaction(s) in 271ms (state_validation=59ms block_commit=119ms state_commit=67ms)
2018-12-28 04:52:12.793 UTC [kvledger] CommitWithPvtData -> INFO 138 [mychannel] Committed block [8] with 1 transaction(s) in 191ms (state_validation=17ms block_commit=142ms state_commit=7ms)
2018-12-28 04:52:31.498 UTC [kvledger] CommitWithPvtData -> INFO 141 [mychannel] Committed block [9] with 1 transaction(s) in 201ms (state_validation=40ms block_commit=130ms state_commit=8ms)
2018-12-28 04:52:49.537 UTC [kvledger] CommitWithPvtData -> INFO 14a [mychannel] Committed block [10] with 1 transaction(s) in 174ms (state_validation=8ms block_commit=133ms state_commit=11ms)
2018-12-28 08:08:21.549 UTC [kvledger] CommitWithPvtData -> INFO 15a [mychannel] Committed block [11] with 1 transaction(s) in 365ms (state_validation=76ms block_commit=159ms state_commit=65ms)
2018-12-28 08:09:27.600 UTC [kvledger] CommitWithPvtData -> INFO 169 [mychannel] Committed block [12] with 1 transaction(s) in 245ms (state_validation=43ms block_commit=124ms state_commit=52ms)
2018-12-28 08:09:39.675 UTC [kvledger] CommitWithPvtData -> INFO 172 [mychannel] Committed block [13] with 1 transaction(s) in 187ms (state_validation=16ms block_commit=119ms state_commit=9ms)
2018-12-28 08:09:45.964 UTC [kvledger] CommitWithPvtData -> INFO 17b [mychannel] Committed block [14] with 1 transaction(s) in 170ms (state_validation=7ms block_commit=124ms state_commit=17ms)
2018-12-28 08:09:49.695 UTC [kvledger] CommitWithPvtData -> INFO 184 [mychannel] Committed block [15] with 1 transaction(s) in 180ms (state_validation=8ms block_commit=127ms state_commit=21ms)
2018-12-28 08:10:19.553 UTC [kvledger] CommitWithPvtData -> INFO 18d [mychannel] Committed block [16] with 1 transaction(s) in 206ms (state_validation=16ms block_commit=125ms state_commit=41ms)
2018-12-28 08:10:43.677 UTC [kvledger] CommitWithPvtData -> INFO 196 [mychannel] Committed block [17] with 1 transaction(s) in 212ms (state_validation=38ms block_commit=120ms state_commit=29ms)
2018-12-28 08:11:01.504 UTC [kvledger] CommitWithPvtData -> INFO 19f [mychannel] Committed block [18] with 1 transaction(s) in 248ms (state_validation=68ms block_commit=149ms state_commit=8ms)
2018-12-29 01:33:52.996 UTC [kvledger] CommitWithPvtData -> INFO 1af [mychannel] Committed block [19] with 1 transaction(s) in 278ms (state_validation=41ms block_commit=124ms state_commit=85ms)
2018-12-29 01:34:49.641 UTC [kvledger] CommitWithPvtData -> INFO 1b8 [mychannel] Committed block [20] with 1 transaction(s) in 206ms (state_validation=46ms block_commit=122ms state_commit=14ms)
2018-12-29 01:34:58.015 UTC [kvledger] CommitWithPvtData -> INFO 1c1 [mychannel] Committed block [21] with 1 transaction(s) in 187ms (state_validation=20ms block_commit=121ms state_commit=23ms)
2018-12-29 01:35:47.656 UTC [kvledger] CommitWithPvtData -> INFO 1d1 [mychannel] Committed block [22] with 1 transaction(s) in 175ms (state_validation=13ms block_commit=119ms state_commit=20ms)
2019-01-03 00:49:01.202 UTC [kvledger] CommitWithPvtData -> INFO 1eb [mychannel] Committed block [23] with 1 transaction(s) in 1227ms (state_validation=261ms block_commit=332ms state_commit=554ms)
2019-01-03 00:56:00.726 UTC [kvledger] CommitWithPvtData -> INFO 1fe [mychannel] Committed block [24] with 1 transaction(s) in 339ms (state_validation=87ms block_commit=132ms state_commit=85ms)
2019-01-03 00:56:21.422 UTC [kvledger] CommitWithPvtData -> INFO 207 [mychannel] Committed block [25] with 1 transaction(s) in 243ms (state_validation=30ms block_commit=136ms state_commit=53ms)
2019-01-03 00:57:14.643 UTC [kvledger] CommitWithPvtData -> INFO 210 [mychannel] Committed block [26] with 1 transaction(s) in 213ms (state_validation=20ms block_commit=132ms state_commit=16ms)
2019-01-03 00:57:21.958 UTC [kvledger] CommitWithPvtData -> INFO 219 [mychannel] Committed block [27] with 1 transaction(s) in 160ms (state_validation=8ms block_commit=118ms state_commit=7ms)
2019-01-03 00:57:57.125 UTC [kvledger] CommitWithPvtData -> INFO 222 [mychannel] Committed block [28] with 1 transaction(s) in 230ms (state_validation=8ms block_commit=168ms state_commit=29ms)
2019-01-03 00:58:01.764 UTC [kvledger] CommitWithPvtData -> INFO 22b [mychannel] Committed block [29] with 1 transaction(s) in 199ms (state_validation=25ms block_commit=132ms state_commit=13ms)
2019-01-03 00:59:21.805 UTC [kvledger] CommitWithPvtData -> INFO 234 [mychannel] Committed block [30] with 1 transaction(s) in 349ms (state_validation=67ms block_commit=156ms state_commit=96ms)
2019-01-03 00:59:55.484 UTC [kvledger] CommitWithPvtData -> INFO 23d [mychannel] Committed block [31] with 1 transaction(s) in 172ms (state_validation=8ms block_commit=115ms state_commit=25ms)
2019-01-03 00:59:59.726 UTC [kvledger] CommitWithPvtData -> INFO 246 [mychannel] Committed block [32] with 1 transaction(s) in 164ms (state_validation=9ms block_commit=123ms state_commit=7ms)
2019-01-03 01:00:36.369 UTC [kvledger] CommitWithPvtData -> INFO 24f [mychannel] Committed block [33] with 1 transaction(s) in 185ms (state_validation=9ms block_commit=115ms state_commit=39ms)
2019-01-03 01:00:40.101 UTC [kvledger] CommitWithPvtData -> INFO 258 [mychannel] Committed block [34] with 1 transaction(s) in 162ms (state_validation=9ms block_commit=120ms state_commit=6ms)
2019-01-03 01:01:06.795 UTC [kvledger] CommitWithPvtData -> INFO 261 [mychannel] Committed block [35] with 1 transaction(s) in 211ms (state_validation=7ms block_commit=119ms state_commit=51ms)
2019-01-03 01:01:09.205 UTC [kvledger] CommitWithPvtData -> INFO 26a [mychannel] Committed block [36] with 1 transaction(s) in 162ms (state_validation=8ms block_commit=123ms state_commit=8ms)
2019-01-03 01:01:30.517 UTC [kvledger] CommitWithPvtData -> INFO 273 [mychannel] Committed block [37] with 1 transaction(s) in 294ms (state_validation=48ms block_commit=152ms state_commit=72ms)
2019-01-03 01:01:36.936 UTC [kvledger] CommitWithPvtData -> INFO 27c [mychannel] Committed block [38] with 1 transaction(s) in 166ms (state_validation=8ms block_commit=116ms state_commit=17ms)
```

   이런 식으로 현재까지 쌓인 블록의 height와 해당 transaction을 알 수 있음.
   -i(대소문자 구분 x), -a(모르겠음), -E(여러 문자열을 |로 구분해 출력하도록)

3. vi에디터(들여쓰기) https://blog.naver.com/kkson50/120073192275 ← 참고
   - 전체 블록지정('vG') 후 'gg=G' 입력하면 gg(처음)~G(끝) =(정렬)
   - 원하는 부분만 블록지정 후 '=' 입력하면 정렬
   - >(들여쓰기), <(내어쓰기) 숫자와 합쳐서도 사용 가능!!
   ex) vG gg=G / v 10> 등

4. docker 컨테이너 다 나가졌을 때?
   - $ docker start $(docker ps -aq)로 재시작 할 것!

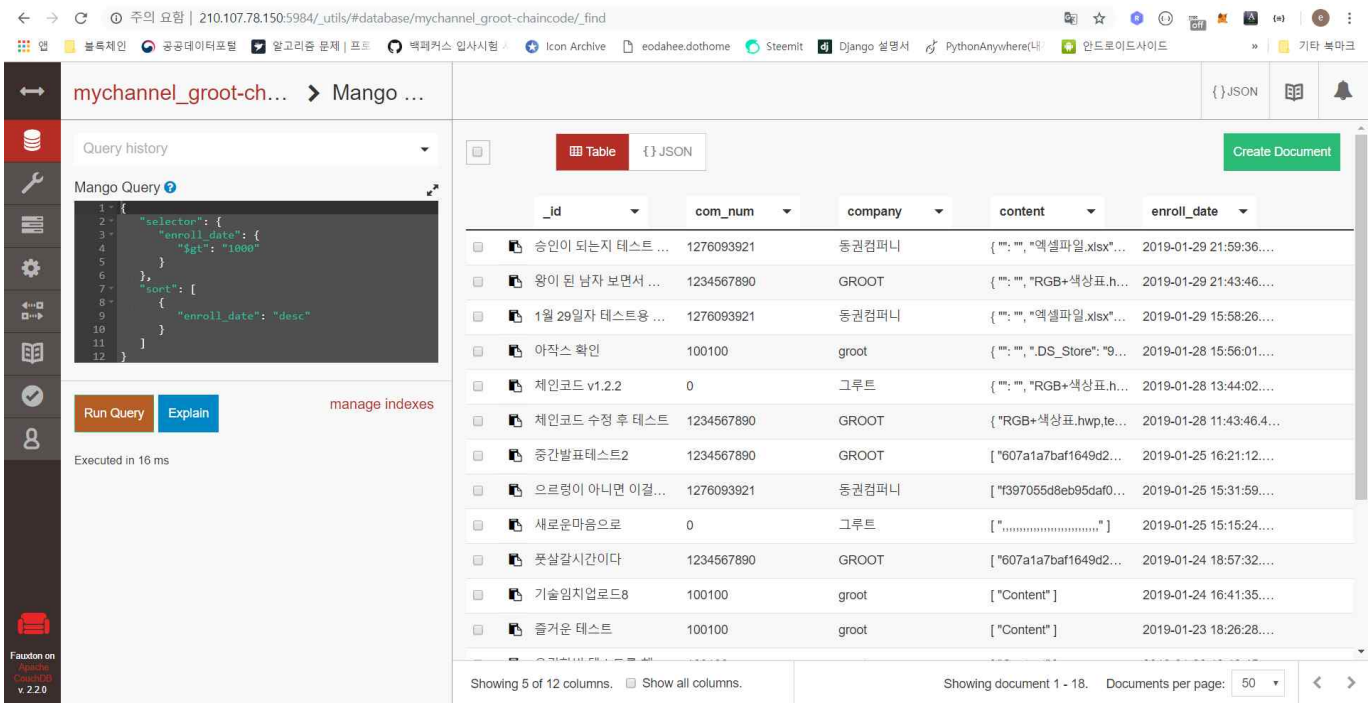
5. 포트 번호 누가 쓰고 있을 때? 에러남
   - $ ps -ef | grep -i node
     작동중인(ps) 모든 프로세스(ef) 중에서 대소문자 구분 없이(i) 'node'라는 단어가 들어가는 놈을 검색하라!!(옵션 설명 알고 싶으면 $ man ps 치면 됨)
     NAME PID PPID 접속시간 소유자 작동시간

```
groot@GROOT-Server4:~/groot/groot-app$ ps -ef | grep -i node
root      2427  2403  1 15:38 ?        00:00:24 peer node start
groot     4833  4805  0 15:53 pts/0    00:00:06 node server.js
groot     5122  5040  0 16:16 pts/1    00:00:00 grep --color=auto -i node
```

   - 실행중인 process 지우면 작동 가능
     $ kill -15 PID (정상종료(-15), 강제종료(-9))
   - $ netstat -na | grep 8001

6. 블록의 hash는 어떻게 얻을 수 있는가?
   - https://godoc.org/github.com/hyperledger/fabric-sdk-go/pkg/client/ledger
   - package ledger의 QueryBlock 함수 이용!
   - 'package ledger query block' 키워드로 검색하자!

7. marbles 예제에서 맨 밑에 블록 나오도록 하는 부분 코드분석
   - 블로그 참고
https://medium.com/wearetheledger/hyperledger-fabric-couchdb-fantastic-queries-and-where-to-find-them-f8a3aecef767
   - GetQueryResult 이용해서 원하는 쿼리문을 couchDB에 전달해 값 출력
     (ex - enroll_data 기준 내림차순 정렬해서 받기 등)



   - constructQueryResponseFromIterator() 함수 복사 # query 결과 print
     getQueryResultForQueryString() 함수 복사 # query string을 입력받아 GetQueryResult에 넣어 결과 받기

8. vi에디터(문자 치환)
   - ex) 문서 전체에 대해 pb를 sc로 치환, 할 때마다 확인받기(c)
     1,$ s/pb/sc/c 또는 % s/pb/sc/c

9. list 안에 있는 json 데이터를 기준으로 정렬하려면 sorted() 함수 이용
   기본값은 오름차순(reverse=False)이므로, 내림차순 정렬을 원하면 reverse=True 입력
   ex) groot_scan = sorted(groot_scan, key=lambda k: k[0].get('Timestamp'), reverse=True)
   → groot_scan의 데이터를 가지고 [0]번 방에 있는 것 중 'Timestamp' 값을 기준으로 내림차순 정렬하겠다.

10. 페이지를 로딩 한 현재시간을 기준으로 경과한 시간 구하기
   - https://godoftyping.wordpress.com/2015/04/19/python-%EB%82%A0%EC%A7%9C-%EC%8B%9C%EA%B0%84%EA%B4%80%EB%A0%A8-%EB%AA%A8%EB%93%88/
   - https://yuddomack.tistory.com/entry/%ED%8C%8C%EC%9D%B4%EC%8D%AC-datetime-%EB%82%A0%EC%A7%9C-%EA%B3%84%EC%82%B0

```python
def groot_transaction(request):
    transaction = get_tx()
    time = datetime.datetime.now()

    for tx in transaction :
        timestamp = tx[0].get("Timestamp")[0:19]
        timestamp = datetime.datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')
        # print(timestamp)
        diff = time - timestamp # 시간차이 구하기
        diff = str(diff)[:-7] # milisecond 제외한 값만 보내기
        tx[0]["Timestamp"] = diff # 값 update

    return render(request, 'groot/groot_transaction.html', {'transactions':transaction})
```

timestamp는 string값이기 때문에 datetime함수에 형식을 지정해서 넣어주면 type 변환
time끼리는 +,-연산이 가능함(뒤에 .000000 milisecond 값 제외하고 web에 값 넘기기)

11. Hyperledger Fabric v1.4 릴리즈 내용 참고
   - https://miiingo.tistory.com/192?category=644184 블로그 설명 보기

● Block 관련

1. fabric-sdk-go 이용
   - https://github.com/hyperledger/fabric-sdk-go/blob/master/pkg/client/ledger/ledger.go#L170
   - 아이디어 : QueryInfo()로 블록 height를 가져와 1~height만큼 QueryBlock() 함수 실행
   - 새로운 체인코드를 만들어 go build시 실행파일이 생기지 않음(오류는 없음)
   - 세부 함수 관련 정보 : https://godoc.org/github.com/hyperledger/fabric-sdk-go/pkg/client/ledger#example-Client-QueryBlock

2. fabric-sdk for node.js 이용
   - https://fabric-sdk-node.github.io/global.html#Block__anchor 블록 구조 숙지
   - https://stackoverflow.com/questions/53518276/hyperledger-fabric-how-to-decode-data-hash-to-return-the-actual-data
   - 아이디어 : Django에서 while문에 break 조건을 걸어 해당 함수의 url 호출
   - 위 코드를 controller.js에 대입했으나 오류 발생

3. https://chainhero.io/2018/06/tutorial-build-blockchain-app-v1-1-0/
   - 설치해서 실행 해보려했으나 make 안 됨.

▶ 굳이 chaincode에 만들지 않아도(1번) controller.js에서 적용 가능(2번)
이미 블록에 저장된 내용을 불러오는 것이기 때문에 2번 사용으로 결정!!

[illegible]
강사님 이메일 확인 후 참고!!

* controller.js에 query_tech()라는 함수를 만들어 queryInfo(), queryBlock() 이용해 한 번에 해결해보려 했으나 실패(res.send를 multiple로 할 수 없음 / callback 함수 사용 등)

* controller.js에 queryInfo() 이용 query_tech() 함수 / queryBlock() 이용 query_block() 함수 생성
  - query_block() 함수에서 원하는 json형태로 block이라는 변수를 생성
  - 다른 함수와 똑같이 채널을 생성하고 peer를 channel에 join 시키기
  - query_tech() 함수에서
    channel.queryInfo()를 return해 block의 height와 이전,현재 hash값 반환
  - query_block() 함수에서 parseInt()를 통해 url을 통해 입력받은 값을 형변환 후 대입 → 블록정보 나옴(뒤에 세부 설명)
  - 최종적으로 res.send()를 통해 web페이지에 출력
  - 이후, DJango에서 두 함수를 통해 원하는 값들을 받아옴.
    query_tech()를 통해 얻은 height를 이용해 for문 안에 query_block() 반복
  - 나온 결과들을 새로운 배열에 append 후 원하는대로 가공!!

* 오류들
  - **error: block number must be a positive integer?**
    queryBlock()함수의 인자 값을 받을 때 반드시 int 타입이어야 가능.
    req로 넘겨받으면 string 값이 기본이기 때문에 형 변환을 해줘야 함.
    그냥 int를 붙이는 것이 아니라 parseInt() 함수를 이용!!(아래 블로그 참고)
    https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/parseInt
  - **for문 안에 배열 이상하게 도는 것?**
    sync문제! 동기통신이 아니라서 그러는 것 => promise를 사용해 해결...

* query_tech() 함수 : queryInfo() 이용 최종 블록의 height와 hash 정보 반환

```
query_tech: function(req, res){
        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);

        Fabric_Client.newDefaultKeyValueStore({ path: store_path
        }).then((state_store) => {
            // assign the store to the fabric client
            fabric_client.setStateStore(state_store);
            var crypto_suite = Fabric_Client.newCryptoSuite();
            // and the crypto store (where the users' keys are kept)
            var crypto_store = Fabric_Client.newCryptoKeyStore({path: store_path});
            crypto_suite.setCryptoKeyStore(crypto_store);
            fabric_client.setCryptoSuite(crypto_suite);

            // get the enrolled user from persistence, this user will sign all requests
            return fabric_client.getUserContext('user1', true);
        }).then((user_from_store) => {
            if (user_from_store && user_from_store.isEnrolled()) {
                console.log('Successfully loaded user1 from persistence');
                member_user = user_from_store;
            } else {
                throw new Error('Failed to get user1.... run registerUser.js');
            }
            return channel.queryInfo();
        }).then((blockinfo) => {
            height = blockinfo.height.low; // queryInfo()해서 나오는 height, currentBlockHash,
            console.log('block height : ' + height);

            res.send(blockinfo)
        }).catch((err) => {
            console.error('Failed to query successfully :: ' + err);
        });
```

- 최종 반환하는 blockinfo에는 height, currentHash, previousHash가 있음
  하지만 bytes 값이라 변환해서 사용해줘야 함.
- https://fabric-sdk-node.github.io/global.html#BlockchainInfo__anchor 참고

* query_block() 함수 : queryBlock() 이용 블록의 다양한 정보 가져옴

```
query_block: function(req, res){
        var fabric_client = new Fabric_Client();
        var key = req.params.number;
        var block = {
            info : {
                        block_number: null,
                        previous_hash: null,
                        data_hash: null,
                        transactions: null
                    },
            data : []
        };

        // 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);

        Fabric_Client.newDefaultKeyValueStore({ path: store_path
        }).then((state_store) => {
            // assign the store to the fabric client
            fabric_client.setStateStore(state_store);
            var crypto_suite = Fabric_Client.newCryptoSuite();
            // and the crypto store (where the users' keys are kept)
            var crypto_store = Fabric_Client.newCryptoKeyStore({path: store_path});
            crypto_suite.setCryptoKeyStore(crypto_store);
            fabric_client.setCryptoSuite(crypto_suite);

            // get the enrolled user from persistence, this user will sign all requests
            return fabric_client.getUserContext('user1', true);
        }).then((user_from_store) => {
            if (user_from_store && user_from_store.isEnrolled()) {
                console.log('Successfully loaded user1 from persistence');
                member_user = user_from_store;
            } else {
                throw new Error('Failed to get user1.... run registerUser.js');
            }

            key = parseInt(key); // js의 ...
            return channel.queryBlock(key);
        }).then((q_block) => {
            var tx = '';

            block.info.block_number = q_block.header.number;
            block.info.previous_hash = q_block.header.previous_hash;
            block.info.data_hash = q_block.header.data_hash;
            block.info.timestamp = q_block.data.data.payload.header.channel_header.timestamp;
            block.info.transactions =  q_block.data.data.length;

            q_block.data.data.forEach(transaction => {
                tx = {
                    Transaction_ID: transaction.payload.header.channel_header.tx_id,
                    Creator_ID: transaction.payload.header.signature_header.creator.Mspid,
                    Timestamp: transaction.payload.header.channel_header.timestamp,
                    Belong_block: transaction.payload.data.actions.payload.action.proposal_re...
                    Data: JSON.stringify(transaction.payload.data)
                    Data: transaction.payload.data
                };
            });
            block.data.push(tx);

            console.log(block);
            res.send(block)
        }).catch((err) => {
            res.send('{"error":"error"}');
            console.error('Failed to query successfully :: ' + err);
        });
},
```

- 원하는 형태의 json object인 block 생성 후 queryBlock() 이용 값 대입
- transaction은 여러개가 있을 수 있으므로 for-each문 사용
- https://fabric-sdk-node.github.io/global.html#Block__anchor 출력값 참고

1. DJango에서 값 받아와 가공할 때 생겼던 의문점
   - query_block()으로 가져오는 정보에 current_hash값은 없음
     data_hash값이 있는데 이는 머클트리를 이용한 데이터의(tx) 해쉬 값으로 원하는 정보가 아님
     views.py에서 새로운 current_hash라는 데이터 생성해 넘겨주기!(get_block 함수 내)

```
for i in range(0, len(groot_bscan)):
    if i == len(groot_bscan)-1:# 마지막 블록에서 멈추기
        break
    groot_bscan[i][0]["current_hash"] = groot_bscan[i+1][0].get('previous_hash')
```

   - 시간 순 정렬 어떻게? Transaction 했을 때처럼 정렬
     블록의 시간은 '2019-02-09T06:24:31.103Z' 이런 식으로 나와서 좀 다르게 파싱

```
timestamp = block_parse.get('data')[0]['Timestamp'][0:-5]
timestamp = timestamp.split('T')
timestamp = ' '.join(timestamp)
timestamp = datetime.datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')
timestamp = timestamp + datetime.timedelta(hours=9)

groot_bscan = sorted(groot_bscan, key=lambda k: k[0].get('timestamp'), reverse=True)
```

   - 이전블록 num과 다음블록 num을 알아야 url 이동을 할 수 있는데
     장고 템플릿{{ }} 태그는 사칙연산을 지원하지 않음.
     views.py에서 계산에서 html로 넘겨주기!(groot_block_detail 함수 내)

```
for i in range(0, len(blocks)):
    if blocks[i][0]["block_number"] == str(height):
        blocks[i][0]["pre_block"] = int(blocks[i][0]["block_number"]) - 1# data 추가(이전블록 넘버)
        blocks[i][0]["next_block"] = int(blocks[i][0]["block_number"]) + 1# data 추가(다음블록 넘버)
        block.append(blocks[i][0])
        break
```

   - query_block()으로 가져오는 값의 양이 너무 크고 복잡해 가공하는데 난항 겪음
     JSON과 배열의 조합으로 이루어짐(최대한 원하는 값은 맨 위로 빼서 씀)
     block 안에 transaction의 정보가 들어있기 때문에 get_cert_verify() 함수가 필요 없을 수도 있을 것 같음(2019-02-12 기준)
   - 최대한 traffic을 덜 발생시키는 방향으로 코딩하려고 노력 중.
     get_cert_verify() 함수를 사용하지 않고, query_block()함수와 query_tech(), query_tx() 함수를 잘 배치해서 코딩함.

- 블록의 크기는 sys라이브러리를 이용(블록 안에 있는 str의 글자 수 계산)
  어떤 타입이든 bytes 크기로 변환시켜주는 getsizeof() 함수를 통해 블록크기 출력

```
block_size = str(sys.getsizeof(data)) + 'bytes' # bytes 크기로 블록 size 구하기
```

- config.yaml 파일에서 지정한 블록 크기 정보

```
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

    # Batch Size: Controls the number of messages batched into a block
    BatchSize:

        # Max Message Count: The maximum number of messages to permit in a batch
        MaxMessageCount: 10

        # Absolute Max Bytes: The absolute maximum number of bytes allowed for
        # the serialized messages in a batch.
        AbsoluteMaxBytes: 99 MB

        # Preferred Max Bytes: The preferred maximum number of bytes allowed for
        # the serialized messages in a batch. A message larger than the preferred
        # max bytes will result in a batch larger than preferred max bytes.
        PreferredMaxBytes: 512 KB
```

* 우리 network에서 지정한 블록 배치타임 및 최대 크기
* 1MB = 1,000KB / 1KB = 1,000bytes / 한글(2bytes) / 영숫자(1bytes)
* 최대: 99MB = 99,000KB = 99,000,000bytes
* 권장: 512KB = 512,000bytes

번 호	변 수 명	타 입	DB 지정 크기
1	Technology	string	char(100)
2	Sort	int	int(50)
3	Company	string	char(100)
4	Com_num	int	int(50)
5	Term	int	int(50)
6	Content	map	char(100) : char(64)
7	Client	map	char(100) : int(50)
8	Enroll_date	string	datetime
9	Status	int	int(1)

* 예상? file 하나 업로드 시 500bytes, 100개 업로드 시 최대 16,900bytes = 17KB
  블록 하나의 크기로 충분함!!

● 기타 Django 관련

1. pagenation css 노가다(.css 파일을 따로 만들어 수정)
2. 작은 이미지는 'fas fa-' class 이용
3. matplot 라이브러리 이용 그래프 그리기
   일자별 tx량을 계산해 꺾은선 그래프로 출력하고자 함.
   x축은 오늘 date를 기준으로 10일치를 배열에 저장
   y축은 DB에 저장된 enroll_date가 x축과 같은 것들을 count해 배열에 저장
   pyplot라이브러리의 plot() 함수를 이용해 그래프 그리기!!
   이미지 경로를 지정해 저장 후 web에서 받아와(static) 사용
   https://zzsza.github.io/development/2018/08/24/data-visualization-in-python/ 기초
   https://datascienceschool.net/view-notebook/d0b1637803754bb083b5722c9f2209d0/ 속성지정
   https://stackoverrun.com/ko/q/2534967 색 지정하는 내용

```
# 첫 화면 그래프 출력을 위한 코드
today = datetime.date.today()
day_x = [today] # 일자(최근 10일)
count_y = 0 # 일자별 tx 수
canvas = {} # 그래프를 그릴 최종 데이터
x, y = [], [] # 배열 초기화
for i in range(1,11) : # 9번 실행(배열에 10개 값 쌓이도록)
    day_x.append(today - datetime.timedelta(days=i))

# 일자별 tx 건수 JSON 배열에 추가
for i in range(0,10) :
    for j in range(0, len(e_date)) :
        if str(day_x[i]) == datetime.datetime.strftime(e_date[j], '%Y-%m-%d') : # 그래프에 출력하고자 하는 날짜와 임치일자가 같으면
            count_y = count_y + 1
    canvas[day_x[i]] = count_y
    count_y = 0 # 데이터가 쌓였기 때문에 초기화
# print(canvas)

for key, val in canvas.items() :
    x.append(str(key)[5:]) # 시간 자르고 day까지만 추가
    y.append(val)

x.reverse() # 날짜순 정렬
y.reverse() # 값도 다시 정렬
fig = plt.figure() # 판 제작
plt.plot(x, y, 'wo-', mfc='#17354c') # white, o모양, -선, maker_face_color(구멍뚫기)
# plt.grid(True) # 눈금표시
fig.patch.set_facecolor('#17354c')
fig.set_size_inches(7.5, 2.6)

ax = fig.add_subplot(1,1,1)
ax.set_yticks([0,2,4,6,8,10]) # y축 눈금지정
ax.tick_params(color='white', labelcolor='white')
ax.spines['top'].set_color('none')
ax.spines['left'].set_color('white')
ax.spines['bottom'].set_color('white')
ax.spines['right'].set_color('none')
fig.savefig(r'C:\\Users\여다희\work_django\groot-django\groot\static\groot_scan.png', facecolor=fig.get_facecolor(), transparent=True)
```

   * import matplotlib.pyplot as plt을 통해 그래프 그리기
   * 우선 그림을 그릴 판(figure)을 먼저 그리고 그 안에(subplot을 이용해 가로x세로 1x1 크기로 1번째) 그릴 그래프 plt.plot로 제작
   * 이후 속성들은 블로그 주소 참고하여 작성(축(spines), 축값(set_yticks, tick_params) 등)
4. 복사하기 clipboard 이용 : $ npm install clipboard --save 또는 아래 블로그 참고
   https://twpower.github.io/81-use-clipboard.js-in-javascript 매우 간단!!