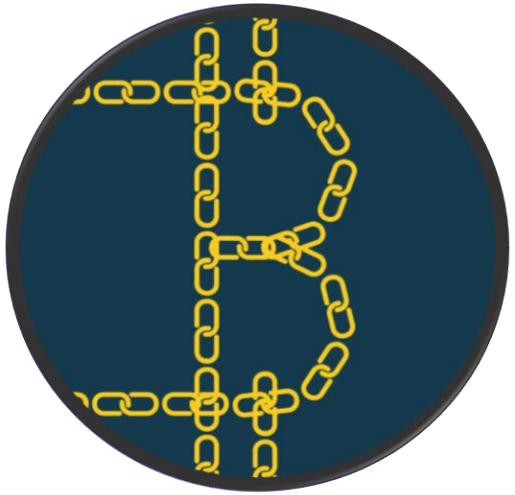


분산컴퓨팅특론

중간 진행상황 보고

Dept. of IT Convergence Engineering
팀원 | 안지용, 최진서, 문기연

전체 일정 진행 현황

▪ Project timeline

- [Google spreadsheet link]

시작일	2023. 4. 1
오늘	2023. 4. 28

일정		시작일	종료일	남은 일자
Blockchain Network Implementation	Status			
개발환경 세팅, 모듈 설계	☑	2023. 4. 10	2023. 4. 14	-
모듈구현, 모듈 테스트(연장)	☐	2023. 4. 17	2023. 5. 5	D-7
중간점검자료 작성	☑	2023. 4. 24	2023. 4. 28	D-Day
피드백 반영, 추가구현	☐	2023. 5. 1	2023. 5. 5	D-7
모듈 통합	☐	2023. 5. 8	2023. 5. 12	D-14
테스트환경 구성, 성능 테스트	☐	2023. 5. 15	2023. 5. 19	D-21
최종 제출자료 작성	☐	2023. 5. 22	2023. 5. 26	D-28

4월																						5월																						
1	4	5	6	7	10	11	12	13	14	17	18	19	20	21	24	25	26	27	28	1	2	3	4	5	8	9	10	11	12	15	16	17	18	19	22	23	24	25	26	29	30	31
토	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수
															연장				연장																									
																			중간점검																									
																			Today																									
																													개발완료														
																																							제출				

▪ 개발현황

- 하드웨어 세팅 → **완료**
- 센싱(DHT11) 모듈 → **구현, 테스트 완료**
- Merkle tree 모듈 → **구현, 테스트 완료**
- 블록체인 자료구조 변경 → **구현, 테스트 완료**
- 블록해시 조작 API → **구현, 테스트 완료**
- 블록체인 네트워크 무결성 감시 모듈 → **구현, 테스트 완료**
- 무결성 복구 모듈 → **구현, 테스트 완료**

역할분배 및 세부 진행현황

▪ Process status

- [Google spreadsheet link]
- 모듈 구현

개발내용	개발자	진행률(%)
센싱(DHT11)모듈 구현	안지용	30
출력(LED)모듈 구현	안지용	100
하드웨어 세팅	안지용	100
User 모듈 구현	문기연	100
Admin 모듈 구현	문기연	100
실험환경 세팅	문기연	10
Merkle tree 구현	최진서	100
블록체인 자료구조 변경	최진서	100
해시조작모듈 구현	최진서	100
네트워크 감시 모듈 구현	최진서	100
일정관리, 협업환경 구축	최진서	상시

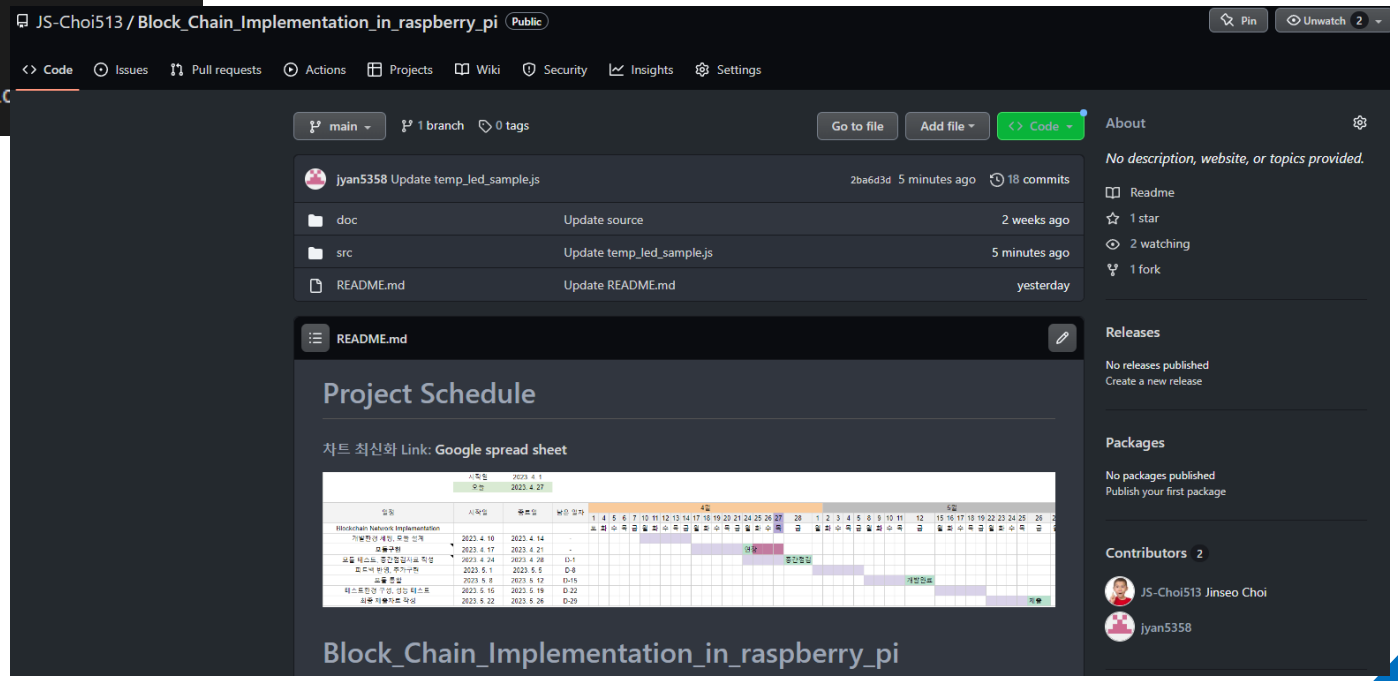
- 모듈 테스트

테스트 내용	개발자	진행률(%)
센싱(DHT11)모듈 테스트	안지용	30
출력(LED)모듈 테스트	안지용	30
User 모듈 테스트	문기연	
Admin 모듈 구현	문기연	
Merkle tree 테스트	최진서	100
블록체인 자료구조 테스트	최진서	100
해시조작모듈 테스트	최진서	100
네트워크 감시 모듈 테스트	최진서	100
협업환경 테스트	최진서	100
모듈 통합 테스트	최진서	0

협업환경

- Github
  - [Github repository link]
  - 하드웨어 세팅 후 SSH 접속
  - 라즈베리파이에 리포지토리를 클론하여 모듈 구현, 테스트 및 버전관리

```
qwe@pi:~/Block_Chain_Implementation_in_raspberry_pi $ ls
doc  README.md  src
qwe@pi:~/Block_Chain_Implementation_in_raspberry_pi $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.9.202.31  netmask 255.255.252.0  broadcast 192.9.203.255
        inet6 fe80::5806:5b85:f4ad:97e5  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:8c:f3:74  txqueuelen 1000  (Ethernet)
        RX packets 655037124  bytes 272484688845 (253.7 GiB)
        RX errors 0  dropped 1504  overruns 0  frame 0
        TX packets 1903470  bytes 285105232 (271.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisio
```



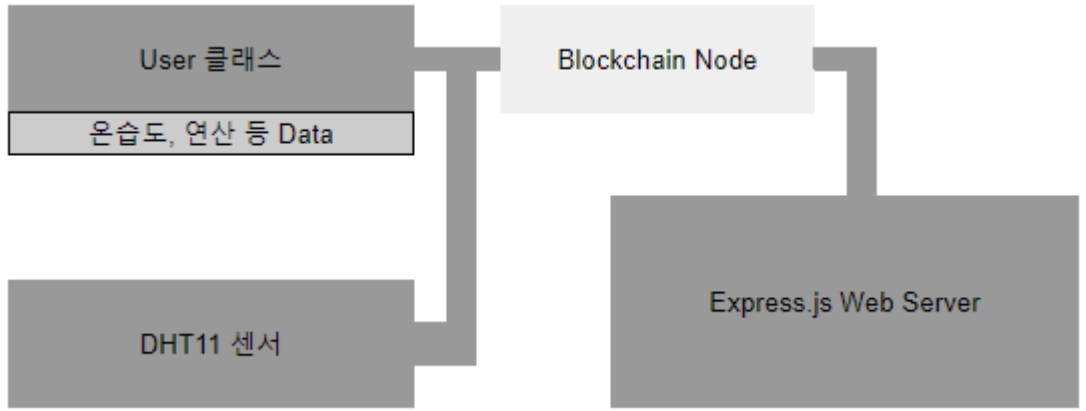
분산컴퓨팅특론 - 중간점검

User 모듈
- 블록체인 연산 요청 처리
- Express.js를 사용하여 웹 서버 구성
- API 엔드포인트 제공

User 클래스 정의
- 블록체인 연산 요청을 처리하는 메소드 구
- requestBlockchainOperation 메소드 사용

Actor	Description
User	블록체인 연산 요청
Admin	노드 등록, 감시

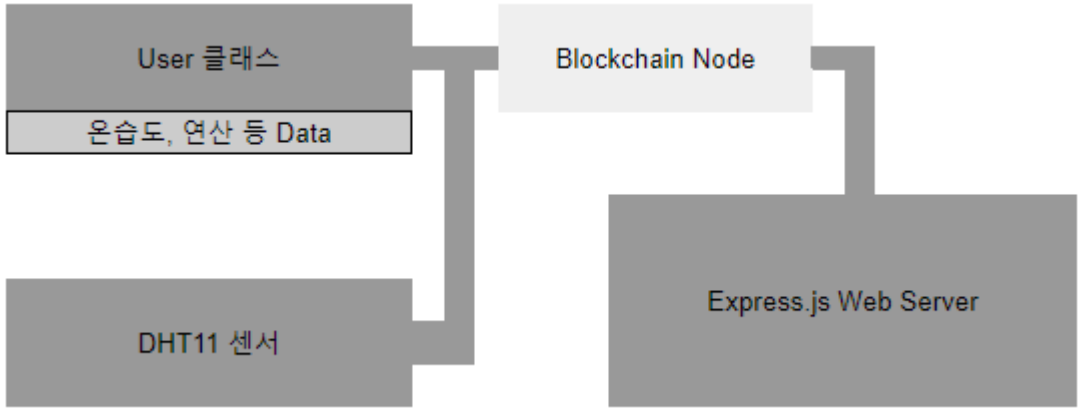
OS	Raspberry Pi OS Lite ARM64
Nodejs	18.15.0
npm	9.6.3
Express.js	4.17.2
Vim	9.0



분산컴퓨팅특론 - 중간점검

requestBlockchainOperation 메소드
- 클라이언트의 블록체인 연산 요청 처리
- 결과 반환 또는 오류 메시지 반환

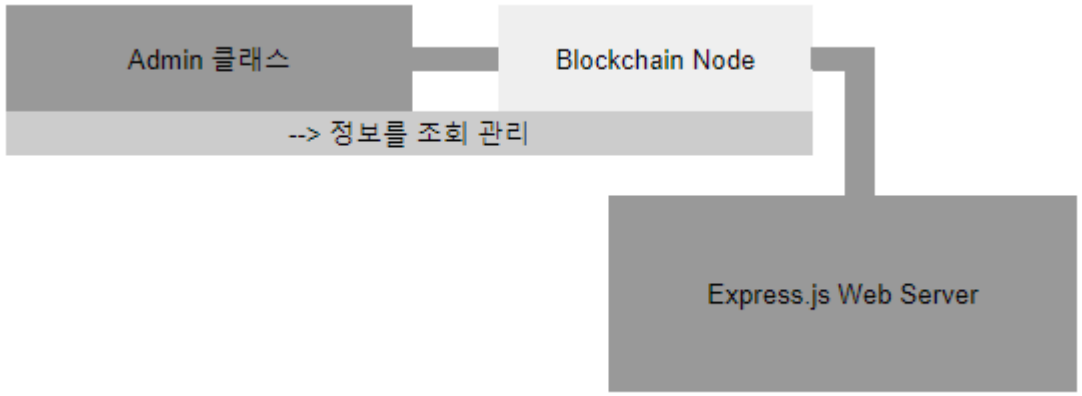
API 엔드포인트 생성 및 웹 서버 시잣
- 블록체인 연산 요청 처리를 위한 API 엔드포인트 생성
- Express.js를 사용하여 웹 서버 구성 및 시작



분산컴퓨팅특론 - 중간점검

Admin 모듈

- 노드 등록 감시 처리
- Express.js를 사용하여 웹 서버 구성
- API 엔드포인트 제공



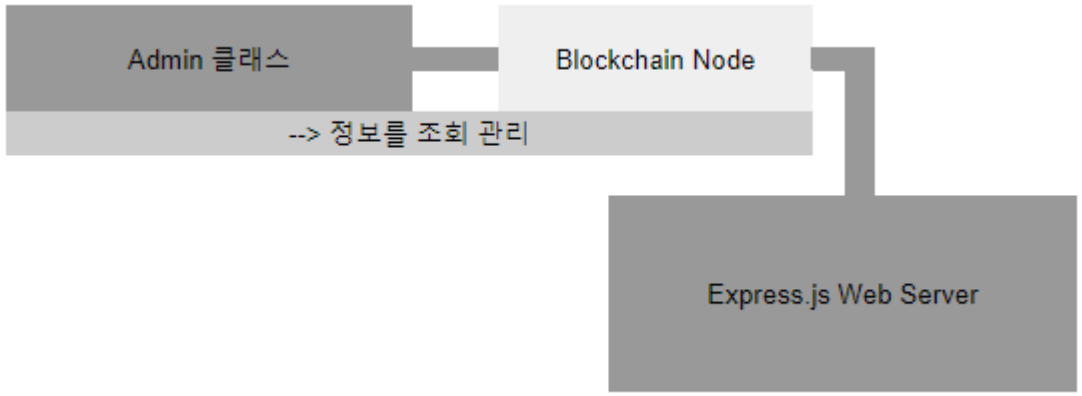
monitorNodeRegistration 메소드

- 노드 등록 상태 확인 및 관리
- 노드 목록 반환 또는 오류 메시지 반환
- 블록체인 시스템에 따라 getRegisteredNodes() 함수를 작성해야 함

분산컴퓨팅특론 - 중간점검

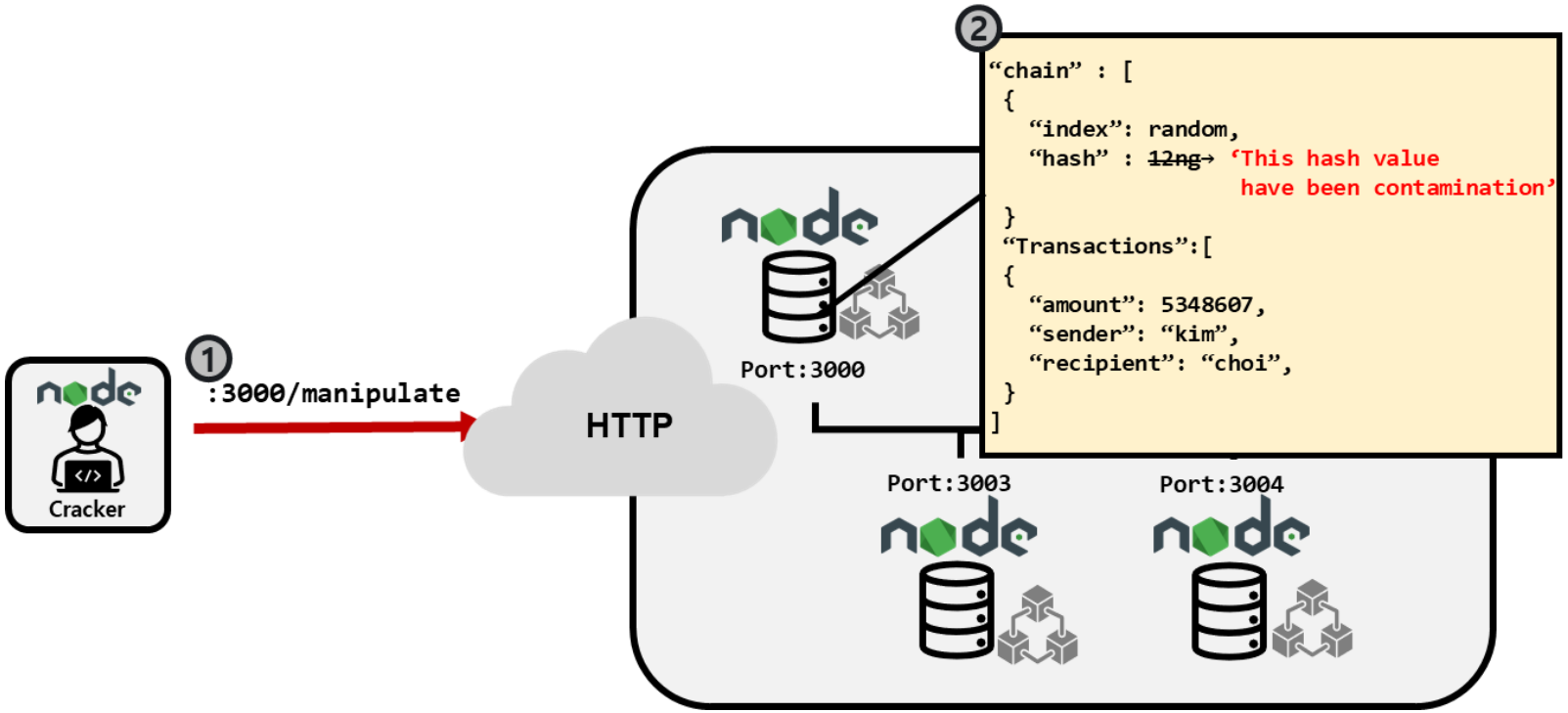
API 엔드포인트 생성 및 웹 서버 시작

- 노드 등록 감시 처리를 위한 API 엔드포인트 생성
- Express.js를 사용하여 웹 서버 구성 및 시작



블록해시 조작모듈

- /manipulate
  - GET API
  - 호출 시 해당 노드 내 블록체인에서 무작위로 1개 블록 해시값 변경



블록해시 조작모듈

- Code
  - networkNode.js

```
+ networkNode.js
 29  * /manipulate
 30  *
 31  * Description: manipulate node block hash value.
 32  *              1. Check block length of current node.
 33  *              2. Randomly select one block
 34  *              3. manipulate block hash  to 'This hash value have been
 35  *              conatmination'.*
 36  */
 37 app.get('/manipulate', function (req, res){
 38   let blk_length =  bitcoin.chain.length;
 39   // 1 ~ block lenght of current chain
 40   let target_blk_idx = Math.floor(Math.random()*(blk_length-2+1)) + 2;
 41   bitcoin.chain[target_blk_idx].hash = 'This hash value have been contamination';
 42   res.send(bitcoin.chain[target_blk_idx].hash);
 43 });
```

현재 노드 체인길이 확인

0: 제네시스 블록 제외

1 ~ last block 중 조작 블록 인덱스를 무작위로 선택

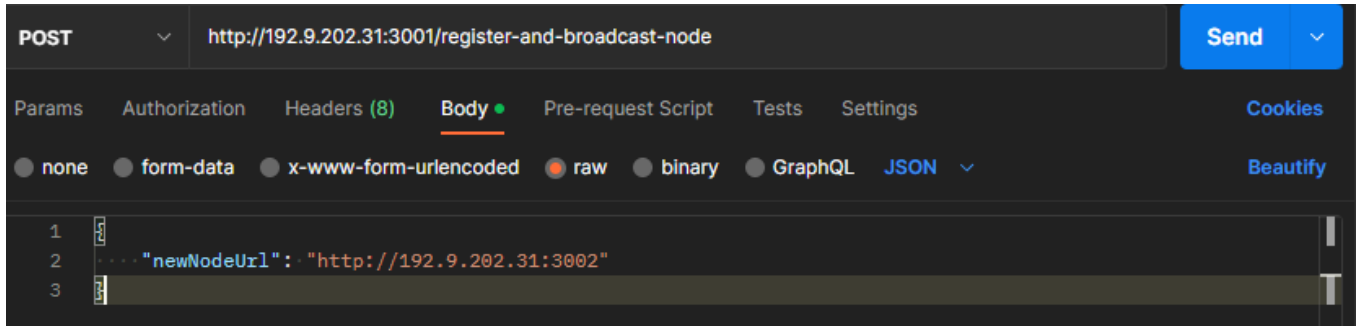
타겟 블록 해시값 변경

블록해시 조작모듈 테스트

- Run blockchain network
  - `npm run test`

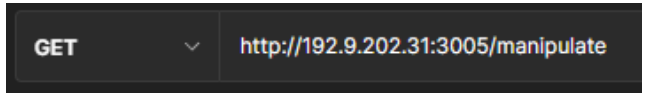
```
[nodemon] watching extensions: js
[nodemon] starting `node dev/networkNode.js 3002 http://192.9.202.31:3002`
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): dev/**/*
[nodemon] watching extensions: js
[nodemon] starting `node dev/networkNode.js 3003 http://192.9.202.31:3003`
[nodemon] 1.19.4
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): dev/**/*
[nodemon] watching extensions: js
[nodemon] starting `node dev/networkNode.js 3005 http://192.9.202.31:3005`
[nodemon] watching dir(s): dev/**/*
[nodemon] watching extensions: js
[nodemon] starting `node dev/networkNode.js 3001 http://192.9.202.31:3001`
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): dev/**/*
[nodemon] watching extensions: js
[nodemon] starting `node dev/networkNode.js 3004 http://192.9.202.31:3004`
Listening on port 3002...
Listening on port 3003...
Listening on port 3005...
Listening on port 3001...
Listening on port 3004...
```

  - 3001 ~ 3005 노드 등록 및 브로드캐스트



블록해시 조작모듈 테스트

- /mine 으로 블록 채굴
  - 6개 블록 채굴
- /manipulate 실행



/manipulate 실행 전


```
{
  "index": 6,
  "timestamp": 1682656342483,
  "transactions": [
    {
      "amount": 12.5,
      "sender": "00",
      "recipient": "e2a9fd10e57c11edabb33d604291bd70",
      "transactionId": "a7a981d0e57d11edabb33d604291bd70"
    }
  ],
  "nonce": 126086,
  "hash": "00002fde131005f590584a689a82eda5505f36be68a0fb785df39bd294240f23",
  "merkle_root": {
    "hash": "22adf50f5fc64c25b01a3196e80742b8f0a2542e15423cafd86443f94d879152",
    "leftNode": null,
    "rightNode": null,
    "parent": null
  },
  "previousBlockHash": "0000a62f304daae3c4dece2ca1dbcca34e11ab5625df396dd97d2afac332c86d"
}
],
"pendingTransactions": [
  {
    "amount": 12.5,
    "sender": "00",
    "recipient": "e2a9fd10e57c11edabb33d604291bd70",
    "transactionId": "ab4211e0e57d11edabb33d604291bd70"
  }
],
"currentNodeUrl": "http://192.9.202.31:3005",
"networkNodes": [
  "http://192.9.202.31:3002",
  "http://192.9.202.31:3003",
  "http://192.9.202.31:3004",
  "http://192.9.202.31:3001"
],
"merkle_tree": "0"
}
```


/manipulate 실행 후


```
{
  "index": 6,
  "timestamp": 1682656342483,
  "transactions": [
    {
      "amount": 12.5,
      "sender": "00",
      "recipient": "e2a9fd10e57c11edabb33d604291bd70",
      "transactionId": "a7a981d0e57d11edabb33d604291bd70"
    }
  ],
  "nonce": 126086,
  "hash": "This hash value have been contamination",
  "merkle_root": {
    "hash": "22adf50f5fc64c25b01a3196e80742b8f0a2542e15423cafd86443f94d879152",
    "leftNode": null,
    "rightNode": null,
    "parent": null
  },
  "previousBlockHash": "0000a62f304daae3c4dece2ca1dbcca34e11ab5625df396dd97d2afac332c86d"
}
],
"pendingTransactions": [
  {
    "amount": 12.5,
    "sender": "00",
    "recipient": "e2a9fd10e57c11edabb33d604291bd70",
    "transactionId": "ab4211e0e57d11edabb33d604291bd70"
  }
],
"currentNodeUrl": "http://192.9.202.31:3005",
"networkNodes": [
  "http://192.9.202.31:3002",
  "http://192.9.202.31:3003",
  "http://192.9.202.31:3004",
  "http://192.9.202.31:3001"
],
"merkle_tree": "0"
}
```

조작 탐지모듈 구현

- 모듈의 필요성
  - 현재 구현되어 있는 ‘가장 긴 체인규칙‘ 기반 합의알고리즘은 현재 네트워크 상에서 임의로 해시가 변경된 블록을 탐지할 수 없음

- 조작 탐지 모듈의 동작
  1. GET API 로 호출
  2. /Check, /Detect, /Recovery, 3개의 API로 구성된
  3. 네트워크 상 존재하는 모든 노드들에 대해 체인 유효성 검사 수행
  4. 검사에서 탐지된 노드 발견 시 해당 노드 복구과정 수행
  5. 노드 복구는 가장 먼저 노드 유효성 검사를 통과한 노드의 블록을 복사

조작 탐지모듈 구현

- Code
  - /check-and-recovery-consistency
  - 해당 API가 내부에서 /detection, /recovery API 수행

```
+ networkNode.js
279 // Function    : /check-and-recovery-consistency
280 /* Description : Current chain consistency check, manipulate detect and recovery
281  *               1. Load chains of other nodes('/blockchain')
282  *               2. Check current chain validation
283  *               3. If current chain invalid, select valid chain from othernode
284  *               4. Replace current chain to valid chain
285 */
286
287 app.get('/check-and-recovery-consistency', function(req, res) {
288     const requestPromises = [];
289     bitcoin.networkNodes.forEach(networkNodeUrl => {
290         const requestOptions = {
291             uri: networkNodeUrl + '/detection',
292             method: 'GET',
293             json: true
294         };
295         requestPromises.push(rp(requestOptions));
296     });
297     Promise.all(requestPromises);
298     console.log('Node consistency check initiate...');
299     res.json({
300         note: 'Node consistency check initiate...'
301     });
302 });
```

호출된 노드를 포함한 네트워크 상 모든 노드에 대해 유효성 검사 요청

메시지 전송

조작 탐지모듈 구현

- Code
  - /detection

```
304 app.get('/detection', function(req, res) {
305     const requestPromises = [];
306     if(!bitcoin.chainIsValid(bitcoin.chain)){
307         console.log('Inconsistency detected ... ');
308         console.log('Node:'+'%s',bitcoin.currentNodeUrl);
309         const requestOptions = {
310                     uri: bitcoin.currentNodeUrl + '/recovery',
311                     method: 'GET',
312                     json: true
313                 };
314         requestPromises.push(rp(requestOptions));
315         Promise.all(requestPromises);
316         res.json({
317             note: 'Chain contemination detected in node('+bitcoin.currentNodeUrl+')
318         });
319     }
320     else{
321         res.json({
322             note: 'This chain is consistency.'
323         });
324     }
325 });
```

현재 노드의 블록체인에 대한 유효성검사

Chain invalid 발생 시 해당 노드로 복구 API 호출

메시지 전송

조작 탐지모듈 구현

- Code
  - /recovery

```
327 app.get('/recovery', function(req, res) {
328     const requestPromises = [];
329     bitcoin.networkNodes.forEach(networkNodeUrl => {
330         const requestOptions = {
331             uri: networkNodeUrl + '/blockchain',
332             method: 'GET',
333             json: true
334         };
335
336         requestPromises.push(rp(requestOptions));
337     });
338     Promise.all(requestPromises)
339     .then(blockchains => {
340         let valid_chain = null;
341         let valid_chain_url = null;
342         blockchains.forEach(blockchain => {
343             if(bitcoin.chainIsValid(blockchain.chain)){
344                 valid_chain = blockchain.chain;
345                 valid_chain_url = blockchain.currentNodeUrl;
346                 return false;
347             }
348         });
349         bitcoin.chain = valid_chain;
350         console.log('Current chain replaced ('+'%s'+'replaced using '+'%s'+')',bitcoin.currentNodeUrl, valid_chain_url);
351         res.json({
352             note: 'This chain have been replaced cause hash manipulate detected.'
353         });
354     });
355 });
```

모든 노드에 대한 블록데이터 로드

블록데이터 유효성 검사 수행
가장 먼저 유효하다고 판단된 블록데이터를
현재 노드 블록데이터로 교체 후 리턴
(중복 유효검사 최소화)

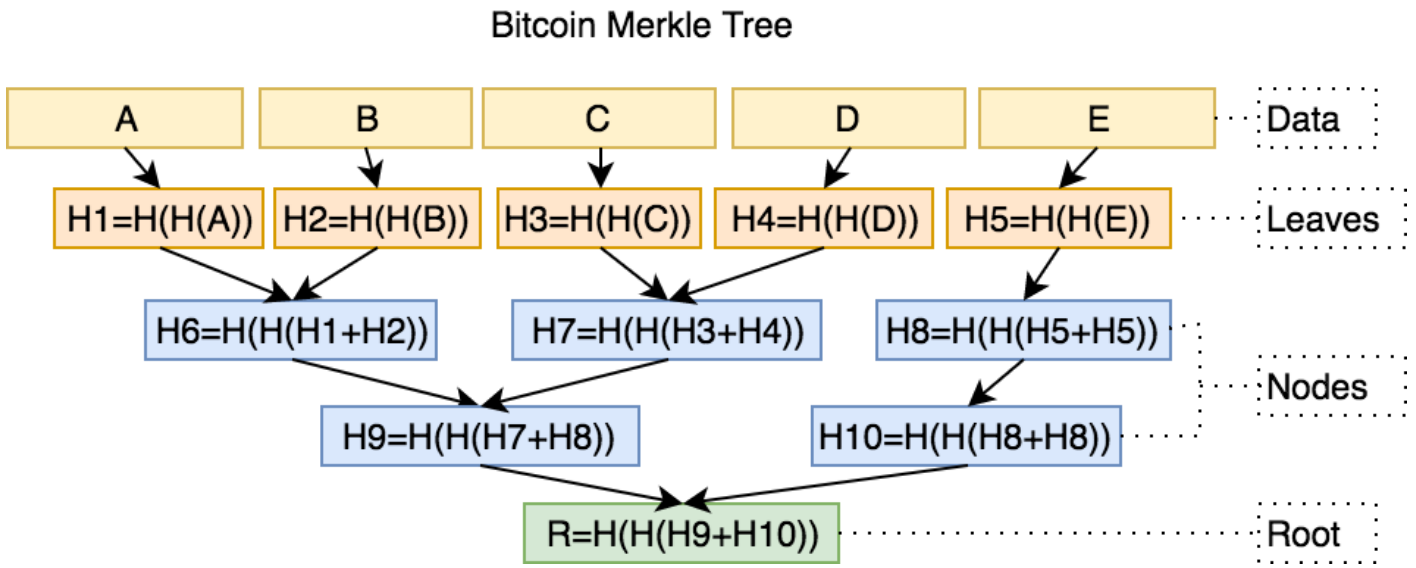
교체된 노드 정보를 메시지로 전송

조작 탐지모듈 테스트

- 3005 노드에서 조작된 해시 값 1개가 존재할 경우

Merkle tree 구현

- Merkle tree 구현방식
  - Merkle_tree.js, Merkle_node.js 모듈로 구성됨
  - 블록생성 시 Pending transaction 리스트를 순회하며 트랜잭션데이터를 노드 객체로 생성하여 한번에 트리를 생성
  - 모든 트랜잭션이 하나의 노드로 생성됨 (1 트랜잭션 : 1노드)
  - 루트 노드 객체는 새로 생성되는 블록 헤더에 추가됨



Merkle tree 구현

- Merkle_tree.js
  - 트리 자료구조, 트리 생성, 트리 연산 구현
  - 트리 자료구조, 루트반환

merkle_tree.js

```
1 const {MerkleNode} = require('./merkle_node')
2 const {Direction, MerkleProofHash} = require('./merkle_proof_hash')
3 const {secureHash} = require('./util')
4
5 class MerkleTree {
6   constructor () {
7     this.rootNode = null
8     this.nodes = []
9     this.leaves = []
10  }
11  getRoot(){
12      const root = this.rootNode;
13      return root;
14  }
```

Merkle tree 구현

- Merkle_tree.js
  - 트리생성
  - 입력으로 노드 리스트를 받음
  - 리스트 크기, 자료형을 확인하여 루트 노드 판단
  - 재귀연산으로 구현

```
28  buildTree (nodes) {
29    if (nodes === undefined || Object.entries(nodes).length == 0){
30        console.log('undefined!');
31        nodes = this.leaves
32    }
33    if (nodes.length === 1|| Object.entries(nodes).length == 0 ){
34        if (Object.entries(nodes).length == 0) this.rootNode = '0';
35        else this.rootNode = nodes[0]
36
37        //console.log('length is 1');
38    }
39    else {
40      const parents = []
41      for (let i = 0; i < nodes.length; i += 2) {// 0, 2, 4, 6
42        //console.log(i);
43          // 1, 3, 5
44        const right = (i + 1 < nodes.length) ? nodes[i + 1] : null
45          // generate parents node i, i+1
46          //          p1 ← 0  1
47          //          p2 ← 2  3
48          //          p3 ← 4  5
49          //         (x) p4 ← 6  null
50          //  ------------ 1Phase------------
51          //          pp1 ← p1(0)  p2(1)
52          //          pp2 ← p3(2)  p4(3)
53          //
54          //  ------------ 2Phase------------
55          //        ppp1 ← pp1  pp2
56          //  ------------ 3Phase------------
57
58        parents.push(new MerkleNode(nodes[i], right))
59      }
60      //console.log('parants lengths is ... %d', parents.length);
61        this.buildTree(parents);
62    }
63  }     생성된 부모노드를 입력으로 재귀호출 루트가 생성될 때 까지 반복
64
65  findLeaf (hash) {
66    return this.leaves.filter(l ⇒ l.hash === hash)[0]
67  }
```

Merkle tree 구현

- Merkle_node.js
  - 각 노드는 고유 해시 값, 형제 노드와 부모 노드에 대한 정보를 가짐
  - 노드객체가 생성될 때 생성자 입력 파라미터에 개수에 따라 노드속성(리프노드, 부모노드)이 변경됨

```
const {secureHash} = require('./util')

class MerkleNode {
  constructor (args) {
    this.hash = 0
    this.leftNode = null
    this.rightNode = null
    this.parent = null
    if (arguments.length === 1 && typeof arguments[0] === 'object') {
      // this is a leaf node
      //console.log(arguments[0]);
      this.hash = secureHash(arguments[0])
      //console.log(this.hash);
    } else {
      // this is a parent node
      this.leftNode = arguments[0]
      this.rightNode = arguments[1] === undefined ? null : arguments[1]
      this.leftNode.parent = this
      if (this.rightNode !== null) this.rightNode.parent = this
      this.computeHash()
    }
  }
```

입력된 노드 길이가 1이고 객체 자료형이 오브젝트(transaction)일 경우

Merkle tree 테스트

- 테스트코드 작성
  - 트랜잭션과 비슷한 형태의 데이터 생성 후 트리생성, 출력 테스트
  - test_mk.js

```
test_mk.js
1 const assert = require('assert')
2 const {MerkleTree} = require('./merkle_tree')
3 const {MerkleNode} = require('./merkle_node')
4 const {secureHash} = require('./util')
5
6
7 const nodes = ['hi', 'there', 'Um', 'Jun', 'Sik'];
8
9  const transaction = {
10     amount: 30,
11     sender: 'asdasdqrqacnfdbc',
12     recipient: 'aqntnvox'
13 };
14 const transaction2 = {
15     amount: 30,
16     sender: 'asddasdqrqacnfdbc',
17     recipient: 'adqntnvox'
18 };
19 const transaction3 = {
20     amount: 30,
21     sender: 'asdasaddqrqacnfdbc',
22     recipient: 'aqntdnvox'
23 };
24 const transaction4 = {
25     amount: 30,
26     sender: 'asdasgdqrqacnfdbc',
27     recipient: 'aqngtnvox'
28 };
```

트랜잭션 형태 데이터 생성: 10개

트랜잭션 데이터가 저장된 리스트 생성

```
const pendingtr = [transaction, transaction2, transaction3,
transaction7, transaction8, transaction9, transaction10];
```

MerkleTree객체 생성

```
const mt = new MerkleTree();
mt.nodes = mt.leaves = pendingtr.map(s => new MerkleNode(s))
console.log(mt.nodes);
```

Map 함수를 사용하여 리스트 내 트랜잭션 데이터를 노드 객체로 생성한 후 트리 내 노드 리스트에 저장

```
const mt = new MerkleTree();
mt.nodes = mt.leaves = pendingtr.map(s => new MerkleNode(s))
console.log(mt.nodes);
mt.buildTree();
//console.log(mt.rootNode.hash);
//mt.appendLeaf('um jun sik');
console.log(mt);
```

노드 정보가 저장된 트리 객체에서 트리생성함수 buildTree() 실행

Merkle tree 테스트

- 테스트 코드 실행결과
  - 트랜잭션과 비슷한 형태의 데이터 생성 후 트리생성, 출력 테스트
  - test_mk.js

```
MerkleTree {
  rootNode: <ref *1> MerkleNode {
    hash: '13f0713ede562281a9862f8a6bba34e3bf6fa63a062c0ba9b1287814821064d6',
    leftNode: MerkleNode {
      hash: '899c0b71887557b10f5746b8d2f6ab92e14ce5e59ed63ea78620afde5b733d20',
      leftNode: [MerkleNode],
      rightNode: [MerkleNode],
      parent: [Circular *1]
    },
    rightNode: MerkleNode {
      hash: '8b32d7f439fc3703a1d38fcd96c4c2a9f5596b97824320e0bd85e03436513187',
      leftNode: [MerkleNode],
      rightNode: [MerkleNode],
      parent: [Circular *1]
    },
    parent: null
  },
  nodes: [
    MerkleNode {
      hash: '19e2f2eb657930563d72c8391075ea71afb35d530398cd34907fa157f175123e',
      leftNode: null,
      rightNode: null,
      parent: [MerkleNode]
    },
    MerkleNode {
      hash: 'd19c07ac1b063ce51d8e1ececee2fc6669c03fe4cc6c6b4ee92f19010d875135',
      leftNode: null,
      rightNode: null,
      parent: [MerkleNode]
    },
    MerkleNode {
      hash: 'ddfe2be7b90668c25423869951d5dad94826c9d9b4e081877a4c3fe19b4478cf',
      leftNode: null,
      rightNode: null,
      parent: [MerkleNode]
    },
```

Blockchain 자료구조 변경

- Blockchin body에 머클루트 추가
- 블록 생성 시 머클트리 생성
- 블록 해시 생성 시 머클루트 데이터 추가

```
const {MerkleTree} = require('./merkle_tree');
const {MerkleNode} = require('./merkle_node');
const {secureHash} = require('./util');

function Blockchain() {
    this.chain = [];
    this.pendingTransactions = [];
    this.currentNodeUrl = currentNodeUrl;
    this.networkNodes = [];
    //how to set genesisblock merkle root?
    this.createNewBlock(100, '0', '0', '0');
};

Blockchain.prototype.createNewBlock = function(nonce, previousBlockHash, hash, mktree) {
    const newBlock = {
        index: this.chain.length + 1,
        timestamp: Date.now(),
        transactions: this.pendingTransactions,
        nonce: nonce,
        hash: hash,
        // merkle_root value
        merkle_root: (mktree == '0') ? '0' : mktree.getRoot(),
        previousBlockHash: previousBlockHash
    };
    //if '/mine' initiate, pending transaction value mapped to merkle tree
    this.merkle_tree = mktree
    this.pendingTransactions = [];
    this.chain.push(newBlock);

    return newBlock;
};
```

제네시스 블록 머클트리 데이터 '0'

Blockchain 자료구조 변경

- 블록 해시생성함수, 자격증명 함수 수정
  - 해시 생성 시 기존 트랜잭션 데이터 대신 머클루트 데이터 추가

```
Blockchain.prototype.hashBlock = function(previousBlockHash, currentBlockData, nonce) {
    //이 부분에서 트랜잭션 조작이 발생하면 해시값이 바뀜
    const dataAsString = previousBlockHash + nonce.toString() + JSON.stringify(currentBlockData);
    const hash = sha256(dataAsString);
    return hash;
};
*/
Blockchain.prototype.hashBlock = function(previousBlockHash, merkle_root, nonce){
    const dataAsString = previousBlockHash + nonce.toString() + merkle_root.hash;
    const hash = sha256(dataAsString);
    return hash;
};

/*
Blockchain.prototype.proofOfWork = function(previousBlockHash, currentBlockDataa) {
    let nonce = 0;
    let hash = this.hashBlock(previousBlockHash, currentBlockData, nonce);
    while (hash.substring(0, 4) !== '0000') {
        nonce++;
        hash = this.hashBlock(previousBlockHash, currentBlockData, nonce);
    }
    return nonce;
};
*/
Blockchain.prototype.proofOfWork = function(previousBlockHash, merkle_root) {
    let nonce = 0;
    let hash = this.hashBlock(previousBlockHash, merkle_root, nonce);
    while (hash.substring(0, 4) !== '0000') {
        nonce++;
        hash = this.hashBlock(previousBlockHash, merkle_root, nonce);
    }
    return nonce;
};
```

Blockchain 자료구조 변경

- 체인 유효성 검사 함수 수정

```
Blockchain.prototype.chainIsValid = function(blockchain) {
    let validChain = true;
    //console.log(blockchain);
    for (var i = 1; i < blockchain.length; i++) {
        const currentBlock = blockchain[i];
        const prevBlock = blockchain[i - 1];
        //const blockHash = this.hashBlock(prevBlock['hash'], { transactions: currentBlock['transactions
index: currentBlock['index'] }, currentBlock['nonce']);
        const blockHash = this.hashBlock(prevBlock['hash'], currentBlock['merkle_root'],
currentBlock['nonce']);

        if (blockHash.substring(0, 4) !== '0000') validChain = false;
        if (currentBlock['previousBlockHash'] !== prevBlock['hash']){
            validChain = false;
            console.log("This chain is invalid..");
            console.log("Current node previous hash: "+"%s",currentBlock['previousBlockHash']);
            console.log("manipulated previous block hash: "+"%s", prevBlock['hash']);
        }
    };

    const genesisBlock = blockchain[0];
    const correctNonce = genesisBlock['nonce'] === 100;
    const correctPreviousBlockHash = genesisBlock['previousBlockHash'] === '0';
    const correctHash = genesisBlock['hash'] === '0';
    const correctTransactions = genesisBlock['transactions'].length === 0;
    //const correctTransactions = genesisBlock['merkle_root'].nodes.length === 0;


    if (!correctNonce || !correctPreviousBlockHash || !correctHash || !correctTransactions) validChain =
false;

    return validChain;
};
```

Blockchain 자료구조 변경후 테스트

- 제네시스 블록 출력
  - Merkle_root 값이 0으로 출력됨

```
// 20230428163615
// http://192.9.202.31:3002/blockchain

{
  "chain": [
    {
      "index": 1,
      "timestamp": 1682667243723,
      "transactions": [

      ],
      "nonce": 100,
      "hash": "0",
      "merkle_root": "0",
      "previousBlockHash": "0"
    },
    {
```

Blockchain 자료구조 변경후 테스트

- 트랜잭션 생성 후 채굴 수행 시
  - 각 블록 별 Merkle_root 해시가 생성된 것을 확인
  - 미결 트랜잭션이 많을 경우 새 블록 생성 후 자식노드가 생성된 것을 확인할 수 있음

<트랜잭션이 1개 일 경우 블록 생성 시>


```
"index": 3,
"timestamp": 1682667688327,
"transactions": [
  {
    "amount": 12.5,
    "sender": "00",
    "recipient": "0a438420e59811edbc77cf614ad834a2",
    "transactionId": "13818b40e59811edbc77cf614ad834a2"
  }
],
"nonce": 32171,
"hash": "0000ab44d8f3dab127e191142051474542f76651f4a61809e04fa51d6f554d26",
"merkle_root": {
  "hash": "2aab8eee2f5edf99b62668cb6c302c59910db30276ab444c6bc0aa7b78b02f4d",
  "leftNode": null,
  "rightNode": null,
  "parent": null
},
"previousBlockHash": "0000dab3fecc115be94b1671f3db6557f7644449c0f8e243394856671b46f831"
},
```


루트 해시확인 , 자식노드가 존재하지 않음

<트랜잭션이 여러 개 일 경우 블록 생성 시>


```
"nonce": 15755,
"merkle_root": {
  "hash": "93898f5128a97b1a85ed97595c482840e0d511b02155e066d7a6c32c3eb5f8f4",
  "leftNode": {
    "hash": "cc22bcff7dd964a338e4c916e7de0776b679b2b4cc3bdc99bf327b979ce8a3e0",
    "leftNode": {
      "hash": "2e00977bf30e334cd9b8d97dd82c4605f59671b20f6c938edff4ba6bcb4c2da9",
      "leftNode": {
        "hash": "22c9dbc26bdd1219fef14408724f3a27005d1c20e1ae423a4a4275a2a77198a7
        "leftNode": null,
        "rightNode": null,
        "parent": "[Circular ~.newBlock.merkle_root.leftNode.leftNode]"
      },
      "rightNode": {
        "hash": "8a370d0392cf7ebcb0dbc779a7464b7047a2c44f0ff19f2a350509b39d170011
        "leftNode": null,
        "rightNode": null,
        "parent": "[Circular ~.newBlock.merkle_root.leftNode.leftNode]"
      },
      "parent": "[Circular ~.newBlock.merkle_root.leftNode]"
    },
    "rightNode": {
      "hash": "d2fca70ce9461af1bb3c6124fd125a4922c133aa0f67938755e3ce8e6dacb3bb",
      "leftNode": {
        "hash": "9f43d488d94f63fff11dc8645c12a1c403465d2480101375792de927053473db
        "leftNode": null,
        "rightNode": null,
        "parent": "[Circular ~.newBlock.merkle_root.leftNode.rightNode]"
      },
      "rightNode": {
        "hash": "3824350f974fead8a9882fa22ed2207e27778639751f3cd3f62bff69871a383f
        "leftNode": null,
        "rightNode": null,
        "parent": "[Circular ~.newBlock.merkle_root.leftNode.rightNode]"
      },
      "parent": "[Circular ~.newBlock.merkle_root.leftNode]"
    },
    "parent": "[Circular ~.newBlock.merkle_root]"
  },
```


자식노드들에 대한 해시정보 확인가능