프로젝트 보고서

팀명: [붐버맨 버니쉬] **팀원:** [이름] ([학번]), [이름] ([학번])

1. 프로젝트 개요

본 프로젝트는 상품의 생산부터 유통, 최종 소비자에게 전달되기까지의 전 과정을 블록체인에 기록하여 데이터의 투명성과 위변조 불가능성을 보장하는 상품 이력 추적 시스템입니다. Express.js를 기반으로 4개의 핵심 라우터(Histories, Mine, Nodes, Transactions)를 구현하였으며, 분산 원장 기술의 핵심인 작업 증명(PoW), 합의 알고리즘(가장 긴 체인 규칙), P2P 노드 간 데이터 동기화 기능을 포함하고 있습니다. 이를 통해 사용자는 특정 상품의 전체 유통 과정을 신뢰할 수 있는 방식으로 조회할 수 있습니다.

2. API 설계

1. Histories 라우터

- **상품 이력 조회 API**

Request

- Method: GET
- URL: /histories/:productId

Request Parameters

Parameter	타입	필수여부	설명
productId	String	필수	조회할 상품의 ID

Response

Response Elements

Element	Depth	값 구분	설명
result	1	Success / Fail	요청 처리 성공 여부
message	1	문자열	결과 메시지
history	1	Array	거래 이력 목록
sender	2	String	(history 배열 내) 발신자 주소
recipient	2	String	(history 배열 내) 수신자 주소
productId	2	String	(history 배열 내) 상품 ID
transactionId	2	String	(history 배열 내) 트랜잭션 ID
timestamp	2	Number	(history 배열 내) 거래 생성 시간

Element	Depth	값 구분	설명
blockIndex	2	Number	(history 배열 내) 포함된 블록 인덱스
blockHash	2	String	(history 배열 내) 포함된 블록 해시

Response JSON 예시

```
{
  "result": "Success",
  "message": "상품 ID [sneakers-123]에 대한 3개의 거래를 찾았습니다.",
  "history": [
    {
      "sender": "Manufacturer",
      "recipient": "Distributor",
      "productId": "sneakers-123",
      "transactionId": "b3c9...",
      "timestamp": 1700000000000,
      "blockIndex": 2,
      "blockHash": "0000a1b2..."
    }
  ]
}
```

2. Mine 라우터

- **블록 채굴 API**

Request

- Method: POST
- URL: /mine

Request Parameters

Parameter	타입	필수여부	설명
minerAddress	String	필수	채굴 보상을 받을 주소

Response

Response Elements

Element	Depth	값 구분	설명
result	1	Success / Fail	채굴 성공 여부
message	1	문자열	결과 메시지
block	1	Object	새로 생성된 블록 정보
index	2	Number	(block 객체 내) 블록 인덱스

Element	Depth	값 구분	설명
timestamp	2	Number	(block 객체 내) 블록 생성 시간
transactions	2	Array	(block 객체 내) 포함된 트랜잭션 목록
nonce	2	Number	(block 객체 내) 작업 증명 결과(Nonce)
hash	2	String	(block 객체 내) 현재 블록 해시
previousBlockHash	2	String	(block 객체 내) 이전 블록 해시

Response JSON 예시

```
{
  "result": "Success",
  "message": "새로운 블록이 성공적으로 채굴되었습니다!",
  "block": {
    "index": 5,
    "timestamp": 1700000500000,
    "transactions": [],
    "nonce": 28391,
    "hash": "0000c3d4...",
    "previousBlockHash": "0000b2c3..."
  }
}
```

3. Nodes 라우터

- **노드 등록 및 브로드캐스트 API**

Request

- Method: POST
- URL: /nodes/register-and-broadcast-node

Request Parameters

Parameter	타입	필수여부	설명
newNodeUrl	String	필수	등록할 새 노드 URL

Response

Response Elements

Element	Depth	값 구분	설명
result	1	Success / Fail	노드 등록 성공 여부
message	1	문자열	결과 메시지

Response JSON 예시

```
{
  "result": "Success",
  "message": "새로운 노드가 성공적으로 네트워크에 등록 및 전파되었습니다."
}
```

4. Transactions 라우터

- **트랜잭션 생성 API**

Request

- Method: POST
- URL: /transactions

Request Parameters

Parameter	타입	필수여부	설명
sender	String	필수	발신자 주소
recipient	String	필수	수신자 주소
productId	String	필수	거래 상품 ID

Response

Response Elements

Element	Depth	값 구분	설명
result	1	Success / Fail	트랜잭션 생성 성공 여부
message	1	문자열	결과 메시지
transaction	1	Object	생성된 트랜잭션 정보
sender	2	String	(transaction 객체 내) 발신자 주소
recipient	2	String	(transaction 객체 내) 수신자 주소
productId	2	String	(transaction 객체 내) 상품 ID
transactionId	2	String	(transaction 객체 내) 트랜잭션 고유 ID
timestamp	2	Number	(transaction 객체 내) 생성 시간

Response JSON 예시

```
{
  "result": "Success",
  "message": "트랜잭션이 생성되어 처리 대기 목록에 추가되었습니다. 다음 블록에 포함될 예정입니
```

```
다.",
  "transaction": {
    "sender": "Alice",
    "recipient": "Bob",
    "productId": "Phone-001",
    "transactionId": "f1e2...",
    "timestamp": 1700001000000
  }
}
```

---

3. 라우터 별 핵심 API 코드

3.1. Histories 라우터 - 상품 이력 조회 API

```
// routes/histories.js

// GET /histories/:productId : 특정 상품의 모든 거래 이력을 조회합니다.
router.get('/:productId', (req, res) => {
    const blockchain = req.app.get('blockchain');
    const { productId } = req.params;

    // Blockchain 클래스의 헬퍼 메소드를 사용하여 비즈니스 로직 캡슐화
    const transactionHistory =
blockchain.getTransactionsByProductId(productId);

    if (transactionHistory.length === 0) {
        return res.status(404).json({
            result: "Fail",
            error: `상품 ID [${productId}]에 대한 거래 이력을 찾을 수 없습니다.`
        });
    }

    res.json({
        result: "Success",
        message: `상품 ID [${productId}]에 대한 ${transactionHistory.length}개
의 거래를 찾았습니다.`,
        history: transactionHistory
    });
});
```

코드 간단 설명: URL 파라미터로 받은 `productId`를 이용하여 블록체인 전체를 순회(헬퍼 메소드 활용)하며 해당 상품과 관련된 모든 트랜잭션을 추출하여 반환합니다.

3.2. Mine 라우터 - 블록 채굴 API

```
// routes/mine.js

router.post('/', async (req, res) => {
```

```
    const blockchain = req.app.get('blockchain');
    const { minerAddress } = req.body;

    if (!minerAddress) return res.status(400).json({/*...*/});

    const lastBlock = blockchain.getLastBlock();
    const previousBlockHash = lastBlock.hash;
    const currentBlockData = {
        transactions: blockchain.pendingTransactions,
        index: lastBlock.index + 1
    };

    // 작업 증명(PoW) 수행
    const nonce = await blockchain.proofOfWork(previousBlockHash,
currentBlockData);
    const blockHash = blockchain.hashBlock(previousBlockHash,
currentBlockData, nonce);

    // 새 블록 생성 및 체인 추가
    const newBlock = blockchain.createNewBlock(nonce, previousBlockHash,
blockHash);

    // *중요* 채굴 보상 트랜잭션은 다음 블록에 포함되도록 블록 생성 후 추가
    blockchain.createNewTransaction('00-REWARD-SYSTEM', minerAddress,
'MINING-REWARD');

    res.json({
        result: "Success",
        message: "새로운 블록이 성공적으로 채굴되었습니다!",
        block: newBlock
    });
});
```

코드 간단 설명: 이전 블록 해시와 현재 대기 중인 트랜잭션 데이터를 기반으로 작업 증명(`proofOfWork`)을 수행하여 유효한 `nonce`를 찾습니다. 이후 새 블록을 생성하여 체인에 연결하고, 채굴자에게 보상을 지급하는 트랜잭션을 다음 블록 대기열에 추가합니다.

3.3. Nodes 라우터 - 합의(Consensus) 알고리즘 API

```
// routes/nodes.js

router.get('/consensus', async (req, res) => {
    const blockchain = req.app.get('blockchain');
    // ... (모든 노드에 블록체인 데이터 요청 로직 생략) ...

    const blockchains = await Promise.all(requestPromises);

    // 가장 긴 유효한 체인 탐색
    for (const response of blockchains) {
        const remoteChain = response.data.chain;
        if (remoteChain.length > maxChainLength &&
```

```
    blockchain.chainIsValid(remoteChain)) {
                maxChainLength = remoteChain.length;
                newLongestChain = remoteChain;
                // ...
            }
        }

        // 내 체인을 더 긴 체인으로 교체
        if (newLongestChain) {
            blockchain.chain = newLongestChain;
            // ...
            res.json({ result: "Success", message: '더 긴 유효 체인을 발견하여 현재 체
인을 교체했습니다.' });
        } else {
            res.json({ result: "Success", message: '현재 체인이 가장 최신 버전이므로 교
체되지 않았습니다.' });
        }
    });
```

코드 간단 설명: 네트워크의 다른 모든 노드에게 블록체인 데이터를 요청하고, '가장 긴 체인 규칙(Longest Chain Rule)'에 따라 자신의 체인보다 길고 유효한 체인이 발견되면 자신의 데이터를 해당 체인으로 동기화합니다.

3.4. Transactions 라우터 - 트랜잭션 생성 API

```
// routes/transactions.js

router.post('/', (req, res) => {
    const blockchain = req.app.get('blockchain');
    const { sender, recipient, productId } = req.body;

    if (!sender || !recipient || !productId) {
        return res.status(400).json({/*...*/});
    }

    const newTransaction = blockchain.createNewTransaction(sender,
recipient, productId);

    res.status(201).json({
        result: "Success",
        message: `트랜잭션이 생성되어 처리 대기 목록에 추가되었습니다. 다음 블록에 포함될 예
정입니다.`,
        transaction: newTransaction
    });
});
```

코드 간단 설명: 발신자, 수신자, 상품 ID를 입력받아 고유 ID를 가진 새로운 트랜잭션 객체를 생성하고, 이를 블록체인의 `pendingTransactions` 배열에 추가합니다.

---

4. 라우터 별 핵심 API 테스트 코드 및 그 결과

4.1. Histories 라우터 테스트

```
// tests/histories.test.js

it('GET /histories/:productId: 특정 상품의 모든 거래 이력을 반환해야 합니다.', async
() => {
    const res = await request(app).get(`/histories/${testProductId}`);
    expect(res.statusCode).toEqual(200);
    expect(res.body.result).toBe('Success');
    expect(res.body.history).toHaveLength(1);
    expect(res.body.history[0].productId).toBe(testProductId);
});
```

결과: PASS tests/histories.test.js

4.2. Mine 라우터 테스트

```
// tests/mine.test.js

it('POST /mine: 새로운 블록을 성공적으로 채굴하고 보상 트랜잭션을 생성해야 합니다.', async
() => {
    // ... (트랜잭션 생성) ...
    const res = await request(app).post('/mine').send({ minerAddress:
minerAddress });

    expect(res.statusCode).toEqual(200);
    expect(res.body.result).toBe('Success');
    expect(blockchain.chain.length).toBe(initialChainLength + 1);
    // 보상 트랜잭션 확인
    expect(blockchain.pendingTransactions).toHaveLength(1);
});
```

결과: PASS tests/mine.test.js

4.3. Nodes 라우터 테스트

```
// tests/nodes.test.js

it('GET /nodes/consensus: 더 긴 유효 체인이 있으면 자신의 체인을 교체해야 합니다.',
async () => {
    // ... (더 긴 체인을 가진 노드 Mocking) ...
    const res = await request(app).get('/nodes/consensus');

    expect(res.statusCode).toEqual(200);
    expect(res.body.message).toContain('더 긴 유효 체인을 발견하여 현재 체인을 교체했
습니다.');
    expect(blockchain.chain.length).toBe(3); // 체인이 교체되었는지 확인
});
```

결과: PASS tests/nodes.test.js

4.4. Transactions 라우터 테스트

```
// tests/transactions.test.js

it('POST /transactions: 새로운 트랜잭션을 생성하여 처리 대기 목록에 추가해야 합니다.',
async () => {
    const res = await request(app)
        .post('/transactions')
        .send({ sender: 'newSender', recipient: 'newRecipient', productId:
'newProduct' });

    expect(res.statusCode).toEqual(201);
    expect(res.body.result).toBe('Success');

expect(blockchain.pendingTransactions).toHaveLength(initialPendingTxCount
+ 1);
});
```

결과: PASS tests/transactions.test.js

부록. 프로젝트 수행 인증 사진

**사진 넣어야함
