2025-1 블록체인 최종보고서

작품명	스마트계약 기반 공동구매 시스템		
팀원	학번	성명	역할/주요업무
	2020810005	김낙균	팀장, 환불 기능 담당
	2020810013	김승혁	조회 기능 담당
	2020810028	도현우	개설 기능 담당
	2019250055	최민성	취소 기능 담당
	2021810057	이지상	참여 기능 담당

2025.6.04

목 차

1. 서　론

1.1. 개발 배경

온라인 쇼핑 환경에서 공동구매는 가격을 절감하고 수요를 예측하는 데 효과적인 방법으로 활용되고 있습니다. 하지만 기존 공동구매 방식은 운영 주체에 대한 신뢰 부족, 중도 이탈로 인한 거래 실패, 환불 처리 지연 등 다양한

문제점이 존재합니다. 이를 해결하기 위해 스마트계약과 블록체인 기술을 접목한 공동구매 시스템을 개발하게 되었습니다.

1.2. 작품 개요 및 목표

사용자는 상품을 등록하고 공동구매를 개설할 수 있으며, 참여자들은 일정 금액의 이더를 전송함으로써 공동구매에 참여할 수 있습니다. 스마트계약은 참여 인원과 기간 조건을 자동으로 확인한 후, 조건이 충족되면 자금이 판매자에게 전달되고, 조건 미충족 시 참여자들에게 자동으로 환불이 이루어집니다. 본 시스템은 공동구매에 필요한 개설, 참여, 조회, 취소, 환불의 전과정을 자동화하여 사용자 편의성과 거래의 신뢰성을 동시에 확보하는 것을 목표로 합니다.

1.3. 작품관련 기반 자료

(기반 되는 소스(링크/교재) 및 설명, 작품에서 추가된 부분 비교 설명)

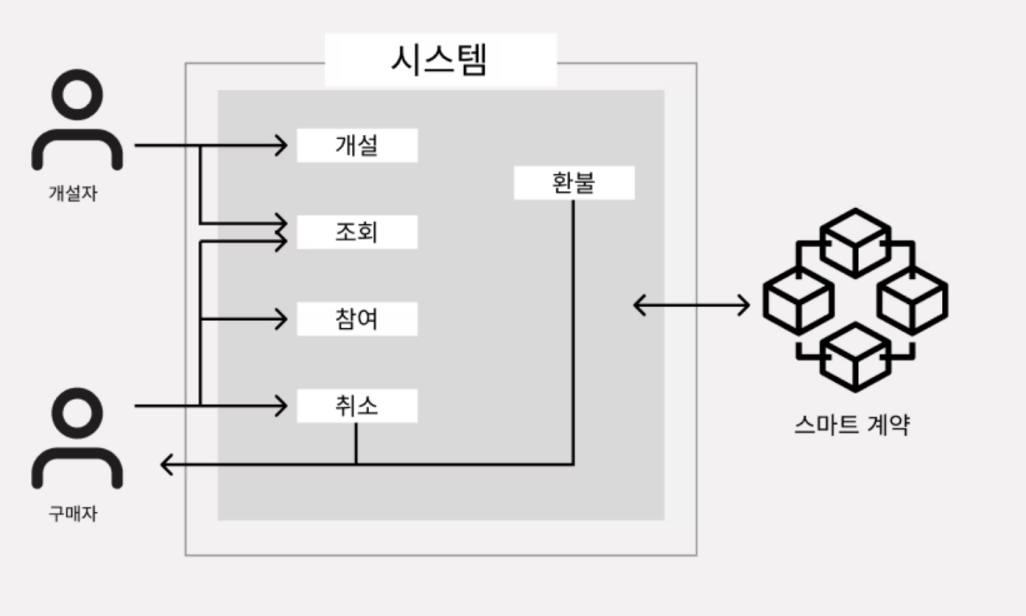
『처음 배우는 블록체인』(가사키 나가토, 시노하라 와타루 저자)의 스마트 계약 예제를 참고하여 Solidity 문법과 기본 구조를 익혔다.
공동구매 참여 인원 조건, 마감 시간, 중복 참여 처리, 환불 및 정산 로직 등 실제 서비스에 필요한 기능을 직접 추가하여 구현하였다.

1.4. 작품의 특징 및 기대효과

① 자동화를 통해 공동구매의 조건 충족 여부를 실시간으로 판단하고 자금 처리를 신속하게 진행할 수 있어 사용자 편의성이 향상됩니다.
② 참여, 취소, 환불 등 주요 절차를 스마트계약이 자동 수행함으로써 운영자의 개입을 최소화하고 오류를 줄일 수 있습니다.
③ 거래 정보가 블록체인에 저장되어 조작이 불가능하며, 참여자 누구나 거래 내역을 확인할 수 있어 **투명성과 신뢰성**을 동시에 확보합니다.
④ 중개 수수료가 없거나 최소화되어 비용 절감 효과가 있으며, 신뢰 기반 없이도 **자율적 거래 환경**이 구현됩니다

2.작품의 분석/설계

2.1설계 개요



2.2기능별 상세설계

기능명	설명
공동구매 개설	개설자가 원하는 상품에 대해 목표 인원 수, 단가, 마감 기한 등을 설정하여 스마트 계약을 생성한다. 생성된 계약은 시스템을 통해 자동으로 블록체인에 배포된다.
공동구매 참여	구매자가 예치금을 충전한 후 원하는 공동구매에 참여할 수 있다. 참여 인원이 목표에 도달하면 계약이 자동으로 완료된다.
공동구매 조회	구매자와 개설자 모두가 현재 개설된 공동구매 목록, 참여 현황, 마감 기한 등의 정보를 실시간으로 확인할 수 있다.
참여 취소	구매자가 마감 전이라면 참여를 취소할 수 있으며, 취소 시 예치금은 자동 환불된다.
환불 처리	공동구매가 실패하거나 마감된 후에도 조건을 만족하지 못한 경우, 개설자 혹은 구매자가 스마트 계약을 통해 환불을 요청할 수 있다.
자금 정산	참여 인원이 목표를 달성하면 개설자가 정산을 요청할 수 있으며, 스마트 계약은 모인 자금을 개설자의 예치금으로 이전시킨다.

3.결론 및 향후과제

3.1작품보완점 및 목표구현 정도

공동구매 개설, 참여, 조회, 취소, 환불 등 주요 기능은 스마트계약으로 구현되었으며, 조건 충족 여부에 따라 자동 정산과 환불도 정상적으로 이루어진다. 간단한 사용자 인터페이스도 함께 구현되어 기능을 직접 확인할 수 있다

3.2개발 환경 및 도구, 버전

개발언어	Solidity
스마트계약 IDE	Remix
버전 정보	Solidity v0.8.x
운영 체제	Windows 11

3.3문제점 및 향후 발전 방향

현재 시스템은 기본 기능 중심으로 구현되어 있어 예외 처리와 보안 강화가 필요하다. UI 역시 기본 수준이므로, 향후 사용자 편의성을 고려한 개선이 요구된다. 메인넷 배포 및 다양한 조건 설정 기능 추가를 통해 실사용에 적합한 형태로 발전할 수 있다.

부 록

R-1. 참고문헌 및 참고사이트

『처음 배우는 블록체인』(가사키 나가토, 시노하라 와타루 저자)의 스마트 계약 예제

```
pragma solidity ^0.8.0;


// SPDX-License-Identifier: UNLICENSED
contract Participationtest {
    struct Purchase {
        uint256 unitPrice;                // 참여 금액
        uint256 targetParticipants;    // 목표 참여자 수
        uint256 currentParticipants;  // 현재 참여자 수
        uint256 deadline;                  // 마감 시간
        bool isCompleted;                  // 완료 여부
        bool isWithdrawn;                  // 인출 여부
        address creator;                   // 개설자
        mapping(address => uint256) participationCount; // 참여 횟수
        address[] participants;          // 참여자 목록 (중복 가능)
    }

    uint256[] public purchaseList;
    mapping(uint256 => Purchase) public purchases;
    mapping(address => uint256) public balances;
    mapping(uint256 => uint256) public purchaseFunds; // 공동구매별 모인 돈
```

```
    event Deposited(address indexed user, uint256 amount);
    event Participated(uint256 indexed purchaseId, address indexed participant, uint256 count);
    event ParticipationCancelled(uint256 indexed purchaseId, address indexed participant, uint256 count);
    event Refunded(uint256 indexed purchaseId, address indexed participant, uint256 amount);
    event RefundedAll(uint256 indexed purchaseId);
    event Withdrawn(uint256 indexed purchaseId, address indexed creator, uint256 amount);

    // 사용자 예치금 충전
    function deposit() external payable {
        require(msg.value > 0, "No ETH sent");
        balances[msg.sender] += msg.value;
        emit Deposited(msg.sender, msg.value);
    }

    // 공동구매 생성
    function setupPurchase(
        uint256 purchaseId,
        uint256 _unitPrice,
        uint256 _targetParticipants,
        uint256 durationSeconds
    ) external {
        Purchase storage p = purchases[purchaseId];
        p.unitPrice = _unitPrice;
        p.targetParticipants = _targetParticipants;
        p.deadline = block.timestamp + durationSeconds;
        p.isCompleted = false;
        p.isWithdrawn = false;
        p.creator = msg.sender;
        purchaseList.push(purchaseId);
    }
```

```
// 참여 (중복 가능)
    function participate(uint256 purchaseId) external {
        Purchase storage p = purchases[purchaseId];
        require(p.deadline != 0, "Purchase does not exist");
        require(block.timestamp <= p.deadline, "Deadline passed");
        require(!p.isCompleted, "Purchase completed");
        require(balances[msg.sender] >= p.unitPrice, "Insufficient balance");

        // 예치금 차감 및 참여 기록
        balances[msg.sender] -= p.unitPrice;
        purchaseFunds[purchaseId] += p.unitPrice; // 공동구매 풀에 추가
        p.participationCount[msg.sender] += 1;
        p.participants.push(msg.sender);
        p.currentParticipants += 1;

        emit Participated(purchaseId, msg.sender, p.participationCount[msg.sender]);

        if (p.currentParticipants >= p.targetParticipants) {
            p.isCompleted = true;
        }
    }

    // 참여 취소 (마감 전 개별 환불)
    function cancelParticipation(uint256 purchaseId) external {
        Purchase storage p = purchases[purchaseId];
        require(p.deadline != 0, "Purchase does not exist");
        require(block.timestamp < p.deadline, "Deadline passed");
        require(!p.isCompleted, "Purchase completed");
        uint256 count = p.participationCount[msg.sender];
        require(count > 0, "No participation to cancel");
```

```
// 제한시간 전후 환불: 마감 전에는 개설자만, 마감 후에는 누구나 일괄 환불
    function refundAll(uint256 purchaseId) external {
        Purchase storage p = purchases[purchaseId];
        require(p.deadline != 0, "Purchase does not exist");
        require(!p.isCompleted, "Purchase completed, cannot refund");

        // 개설자는 언제든 환불 가능, 다른 사용자는 마감 기한 이후에만 가능
        require(
            msg.sender == p.creator || (block.timestamp > p.deadline),
            "Only creator before deadline, anyone after deadline"
        );

        // 중복 환불 방지를 위해 고유한 참여자들만 처리
        address[] memory uniqueParticipants = new address[](p.participants.length);
        uint256 uniqueCount = 0;

        // 고유한 참여자들 추출
        for (uint256 i = 0; i < p.participants.length; i++) {
            address user = p.participants[i];
            bool alreadyProcessed = false;

            // 이미 처리된 사용자인지 확인
            for (uint256 j = 0; j < uniqueCount; j++) {
                if (uniqueParticipants[j] == user) {
                    alreadyProcessed = true;
                    break;
                }
            }
        }
```

```
// 새로운 사용자면 추가
                if (!alreadyProcessed) {
                    uniqueParticipants[uniqueCount] = user;
                    uniqueCount++;
                }
            }

            // 고유한 참여자들에게만 환불 처리
            for (uint256 i = 0; i < uniqueCount; i++) {
                address user = uniqueParticipants[i];
                uint256 refundAmount = p.participationCount[user] * p.unitPrice;
                if (refundAmount > 0) {
                    balances[user] += refundAmount;
                    emit Refunded(purchaseId, user, refundAmount);
                    p.participationCount[user] = 0;
                }
            }

            // 참여자 목록과 참여자 수 초기화
            delete p.participants;
            p.currentParticipants = 0;
            purchaseFunds[purchaseId] = 0; // 공동구매 풀 초기화

            emit RefundedAll(purchaseId);
        }

        // 목표 달성 시 개설자가 정산받는 함수
        function withdraw(uint256 purchaseId) external {
            Purchase storage p = purchases[purchaseId];
            require(msg.sender == p.creator, "Only creator can withdraw");
            require(p.isCompleted, "Purchase not completed yet");
            require(!p.isWithdrawn, "Already withdrawn");
```

```
        uint256 totalAmount = purchaseFunds[purchaseId];
        require(totalAmount > 0, "No funds to withdraw");

        p.isWithdrawn = true;
        purchaseFunds[purchaseId] = 0; // 인출 후 초기화

        // 개설자의 잔액에 추가 (직접 ETH 전송 대신)
        balances[p.creator] += totalAmount;

        emit Withdrawn(purchaseId, msg.sender, totalAmount);
    }
// 전체 공동구매 목록 조회
    function getPurchaseInfo() public view returns (
        uint256[] memory ids,
        uint256[] memory unitPrices,
        uint256[] memory targetCounts,
        uint256[] memory currentCounts,
        uint256[] memory deadlines,
        bool[]    memory completeds
    ) {
        uint256 len = purchaseList.length;
        ids = new uint256[](len);
        unitPrices = new uint256[](len);
        targetCounts = new uint256[](len);
        currentCounts = new uint256[](len);
        deadlines = new uint256[](len);
        completeds = new bool[](len);

        for (uint256 i = 0; i < len; i++) {
            uint256 pid = purchaseList[i];
            Purchase storage p = purchases[pid];
            ids[i] = pid;
            unitPrices[i] = p.unitPrice;
```

```
targetCounts[i] = p.targetParticipants;
                currentCounts[i] = p.currentParticipants;
                deadlines[i] = p.deadline;
                completeds[i] = p.isCompleted;
            }
        }

        // 참여 횟수 조회
        function getParticipationCount(uint256 purchaseId, address user) external view returns (uint256)
{
            return purchases[purchaseId].participationCount[user];
        }

        // 공동구매별 모인 금액 조회
        function getPurchaseFunds(uint256 purchaseId) external view returns (uint256) {
            return purchaseFunds[purchaseId];
        }
}
```