2025-1 블록체인 최종보고서

작품명	NFT 기반 공연 티켓팅 웹서비스		
팀원	학번	성명	역할/주요업무
	2019250032	육창민	팀원/백엔드
	2020810012	김승권	팀장/백엔드
	2023810089	구찬서	팀원/백엔드

목 차

1. 서 론

1.1. 개발 배경(주제 선정 이유)

강의로 배운 내용을 실제 개발에 적용해보기 위해 여러 가지 적용사례를 찾던 도중 최근 대두되고 있는 암표피해 사례 기사를 접하게 되어 이 개발 주제로 정하게 되었다.

1.2. 작품 개요 및 목표

기존 티켓팅 시스템의 문제점을 개선하고 신뢰성을 높일 수 있다. 티켓 거래 데이터의 위변조 방지, 투명성 확보, 불공정 행위 억제 등을 통해 사용자에게 더 안전하고 편리한 티켓팅 경험을 제공 할 수 있다.

1.3. 작품관련 기반 자료 (기반 되는 소스(링크/교재) 및 설명, 작품에서 추가된 부분 비교 설명)

처음 배우는 블록체인-비트코인, 이더리움, 솔리디티, 트러플 프레임워크로 배우는 블록체인 이론과 개발

1.4. 작품의 특징 및 기대효과

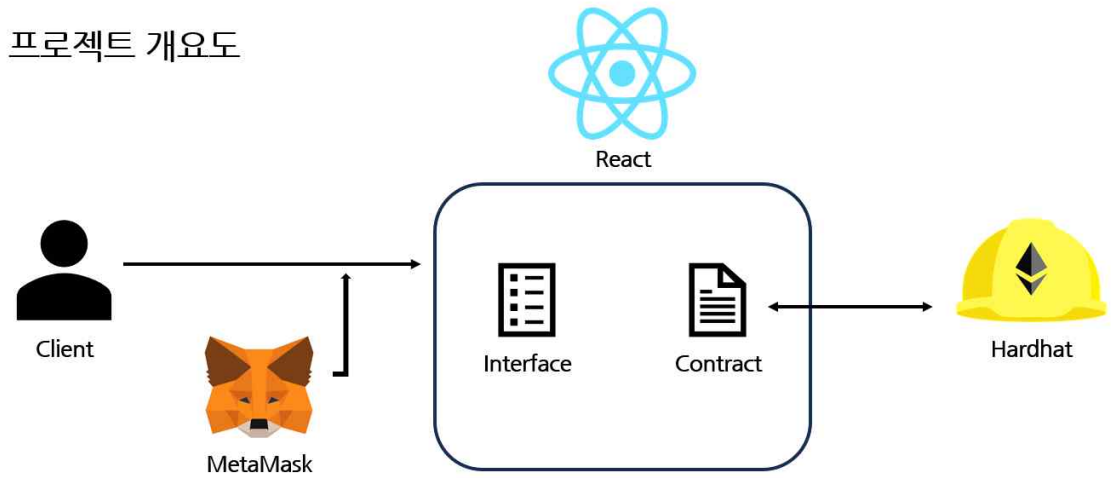
블록체인 티켓팅 개발은 암표 거래를 방지하고, 티켓의 진위 및 소유권을 확실하게 보장하여 티켓팅 시장의 신뢰도를 높일 수 있음. 또한, 스마트 컨트랙트를 활용하여 조건부 거래를 가능하게 하고, 중개 수수료 감소를 통한 티켓 가격 인하도 기대할 수 있음.

1.5. 팀원의 구성 및 역할 분담

팀원	역할 분담
육창민	시스템 개발, 보고서 작성
김승권	시스템 개발, 보고서 작성
구찬서	시스템 개발, 보고서 작성

2. 작품의 분석/설계

2.1. 설계 개요(개념설계, 시스템 구성도)



2.2. 기능별 상세설계

기능명	기능 설명
티켓 구매	- 구매자는 티켓을 구매할 수 있다.
티켓 조회	- 구매자는 본인 소유의 티켓을 조회할 수 있다.
예매 취소	- 구매자는 본인의 예매를 취소할 수 있다.
거래 조회	- 판매자는 티켓의 거래 통계를 조회할 수 있다.

3. 결론 및 향후과제

3.1. 작품보완점 및 목표구현 정도

관리자 계정에서의 플랫폼 관리를 제외하면 최초 설계 기능 구현은 완료.

3.2. 개발 환경 및 도구, 버전

Ethereum wallet: MetaMask
Solidity: v0.8.0
Hardhat
Remix

3.3. 문제점 및 향후 발전 방향

문제점은 크게 기술적 어려움, 규제 문제, 사용자 경험 개선으로 나눌 수 있음.

-기술적 문제

스케일링 문제로 블록체인 트랜잭션 처리 속도가 느리고, 대규모 티켓팅 시스템에 적용하기에는 어려움이 있음.

-규제 문제

가짜 티켓 문제로 블록체인 기술을 악용하여 가짜 티켓을 생성할 가능성이 존재.

- 사용자 경험 개선

블록체인 티켓팅 시스템의 인터페이스가 복잡하고, 사용자가 쉽게 이해하기 어렵다면 사용자 경험이 저해될 수 있습니다.

부 록

R-1. 참고문헌 및 참고사이트

-관련기사

https://www.yna.co.kr/view/AKR20240202056400797

https://www.joongang.co.kr/article/25230022

R-2. 프로그램 소스(각 기능별/모듈별 파일명 설명)

NFTTicketing.sol (NFT토큰 발행 및 반환)

```
// 컴파일러 버전 지정
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract NFTTicketing is ERC721Enumerable, ERC721URIStorage {
    constructor() ERC721("EventTicket", "ETK") {}

    // 필수 함수 오버라이드
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId,
        uint256 batchSize
    ) internal override(ERC721, ERC721Enumerable) {
        super._beforeTokenTransfer(from, to, tokenId, batchSize);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721URIStorage, ERC721Enumerable)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }

    function _burn(uint256 tokenId)
        internal
        override(ERC721, ERC721URIStorage)
    {
        super._burn(tokenId);
    }

    function tokenURI(uint256 tokenId)
        public
        view
        override(ERC721, ERC721URIStorage)
        returns (string memory)
    {
```

```
        return super.tokenURI(tokenId);
    }


    // 본문 시작
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    mapping(address => string) public ticketCodes;
    mapping(address => uint256) public nftIds;



    // 티켓 발행
    function getNFTTicket(string memory ticketCode) public returns (bytes32) {
        ticketCodes[msg.sender] = ticketCode;
        return mintTicket(ticketCode);
    }


    // 랜덤 문자열 + 현재 시간 조합 후 해시 처리해 URI 생성
    function mintTicket(string memory randomString) private returns (bytes32) {
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();

        // 클릭 시점의 timestamp 사용
        uint256 timestamp = block.timestamp;

        // 고유 식별 문자열 구성
        string memory input = string(abi.encodePacked(randomString,
    Strings.toString(timestamp)));
        bytes32 hashedInput = keccak256(abi.encodePacked(input));

        // 해시를 URI로 저장
        string memory fakeURI = Strings.toHexString(uint256(hashedInput), 32);

        _mint(msg.sender, newItemId);
        _setTokenURI(newItemId, fakeURI);

        nftIds[msg.sender] = newItemId;

        return hashedInput;
    }

    // NFT burn
    function burnTicket(string memory ticketCode) public {
        require(nftIds[msg.sender] != 0, unicode"NFT 토큰을 소유하고 있지 않습니다.");
        require(keccak256(abi.encodePacked(ticketCode))                                    ==
    keccak256(abi.encodePacked(ticketCodes[msg.sender])), unicode"티켓 코드가 맞지 않습니다.");

        _burn(nftIds[msg.sender]);

        ticketCodes[msg.sender] = "";
        nftIds[msg.sender] = 0;
    }
}
```

TicketingSystem.sol (기본 기능 구현 / 구매, 조회, 환불)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./NFTTicketing.sol";

contract TicketingSystem is NFTTicketing {
    address public owner;
    uint public ticketPrice;
    //100000000000000000 테스트
    uint public totalTickets;
    uint public ticketsSold;
    uint public saleEndTime;
    string public showName;

    mapping(address => uint) public ticketsOwned;
    mapping(address => uint) public seatNumber;
    mapping(uint => bool) public isSeatTaken;

    event TicketPurchased(address indexed buyer, uint seat);
    event TicketCancelled(address indexed buyer, uint seat);

    constructor(uint _price, uint _totalTickets, uint _durationMinutes, string memory _name) {
        owner = msg.sender;
        ticketPrice = _price;
        totalTickets = _totalTickets;
        saleEndTime = block.timestamp + (_durationMinutes * 1 minutes);
        showName = _name;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, unicode"관리자만 실행할 수 있습니다.");
        _;
    }

    modifier saleActive() {
        require(block.timestamp <= saleEndTime, unicode"판매가 종료되었습니다.");
        _;
    }

    function buyTicket(uint _seat, string memory userInputString) public payable saleActive {
        require(ticketsOwned[msg.sender] == 0, unicode"이미 티켓을 구매하셨습니다.");
        require(_seat >= 1 && _seat <= totalTickets, unicode"유효하지 않은 좌석 번호입니
다.");
        require(!isSeatTaken[_seat], unicode"이미 선택된 좌석입니다.");
        require(msg.value == ticketPrice, unicode"지불 금액이 맞지 않습니다.");

        getNFTTicket(userInputString);
        ticketsOwned[msg.sender] = 1;
        seatNumber[msg.sender] = _seat;
        isSeatTaken[_seat] = true;
        ticketsSold += 1;

        emit TicketPurchased(msg.sender, _seat);
```

```
    }

    function cancelTicket(string memory ticketCode) public {
        require(ticketsOwned[msg.sender] == 1, unicode"보유한 티켓이 없습니다.");

        uint seat = seatNumber[msg.sender];

        burnTicket(ticketCode);
        ticketsOwned[msg.sender] = 0;
        delete seatNumber[msg.sender];
        isSeatTaken[seat] = false;
        ticketsSold -= 1;

        payable(msg.sender).transfer(ticketPrice);

        emit TicketCancelled(msg.sender, seat);
    }

    function getMySeat() public view returns (uint) {
        return seatNumber[msg.sender];
    }

    function getUserSeat(address user) public view returns (uint) {
        return seatNumber[user];
    }

    function getTakenSeats() public view returns (uint[] memory) {
        uint[] memory result = new uint[](ticketsSold);
        uint index = 0;
        for (uint i = 1; i <= totalTickets; i++) {
            if (isSeatTaken[i]) {
                result[index] = i;
                index++;
            }
        }
        return result;
    }

    function getAvailableSeats() public view returns (uint[] memory) {
        uint availableCount = totalTickets - ticketsSold;
        uint[] memory result = new uint[](availableCount);
        uint index = 0;
        for (uint i = 1; i <= totalTickets; i++) {
            if (!isSeatTaken[i]) {
                result[index] = i;
                index++;
            }
        }
        return result;
    }

    function withdraw() public onlyOwner {
        require(address(this).balance > 0, unicode"인출할 잔액이 없습니다.");
        payable(owner).transfer(address(this).balance);
    }
}
```