НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

БЛОКЧЕЙН ТА ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ

Комп'ютерний практикум №2

Реалізація смарт-контракту або анонімної криптовалюти.

Варіант 2.3

Виконали:
Волинець Сергій ФІ-42мн
Сковрон Роман ФІ-42мн
Молдован Дмитро ФІ-42мн

Київ — 2025

1 Мета

Отримання навичок роботи зі смартконтрактами або анонімними криптовалютами.

2 Завдання на лабораторну роботу

Розробка власного смартконтракту.

3 Розробка

Існує багато варіантів розробки смарт контрактів на ethereum. Це, наприклад онлайн Remix IDE, Hardhat, Truffle Suite, Foundry та інші. Ми зупинилися на першому варіанті, бо Remix IDE доволі зручна та юзер-френдлі. Реалізувати будемо систему голосування з назвою `Voting`. Готові приклади можна знайти на наступних сайтах [God21, sol] Даний приклад є лише зразком можливостей смартконтрактів, тому містять елементи, які не варто допускати до релізу. Весь програмний код можна знайти наприкінці (A). Для початку визначимо структуру кандидатів, допоміжні змінні, та конструктор. В даній системі будуть користувачі з правом одного голосу (один голос на довільний публічний ключ), та адмін, що буде оперувати процесом.

```
pragma solidity ^0.6.6;

contract Voting{
    struct Candidate{
        uint id;
        string name;
        uint voteCount;
    }

    address public admin;
    mapping (uint => Candidate) public candidates;
    uint public candidatecount;
    mapping (address => bool) private voter;
    bool public voting_started;
    bool public voting_finished;

    constructor() public{
        admin = msg.sender;
        voting_started = false;
        voting_finished = false;
        addCandidate("Godlin");
        addCandidate("Hilda");
    }
```

```
    function addCandidate(string memory _name) private{
        candidatecount++;
        candidates[candidatecount] = Candidate(candidatecount, _name, 0);
    }

    ...
}
```

Для зручності, в solidity, можна визначати користувацькі модифікатори. Наприклад:

```
    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can perform this");
        _;
    }

    modifier beeforeVoting() {
        require(!voting_started, "Voting not started yet");
        _;
    }
```

Таким чином, можна обмежити використання деяких функцій. Наприклад, адмін може починати та завершувати голосування.

```
    function startVoting() public onlyAdmin beeforeVoting {...}
    function finishVoting() public onlyAdmin {...}
    function continueVoting() public onlyAdmin {...}
```

Для роботи голосування, запропонуємо наступні методи:

```
    function vote(uint _candidateid) public{...}
    function proposeCandidate(string memory _name) public{...}
    function getCurrentWinner()
        public view returns (uint[] memory winnerIndices)
        {...}
```

В solidity немає способу отримати випадкове число, тому потрібно якось викручуватися. Наприклад, можна обчислити випадкове значення, як геш деяких параметрів. Як приклад наведемо наступну функцію.

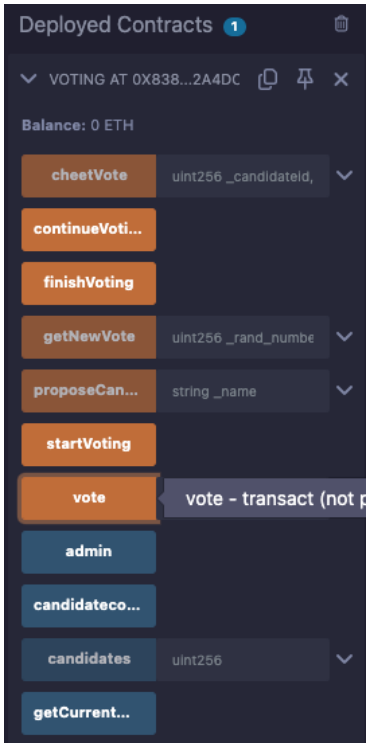
```
    function getNewVote(uint _rand_number) public {
        require(voter[msg.sender]);
        require(voting_started);
        require(!voting_finished, "Voting is finished");


        uint random = uint8(
            uint256(keccak256(abi.encodePacked(
                        block.timestamp,
```

```
                    block.difficulty
            ))) % 10
        ) + 1;
        require(random == _rand_number, "Guesed wrong");

        voter[msg.sender] = false;

    }
```

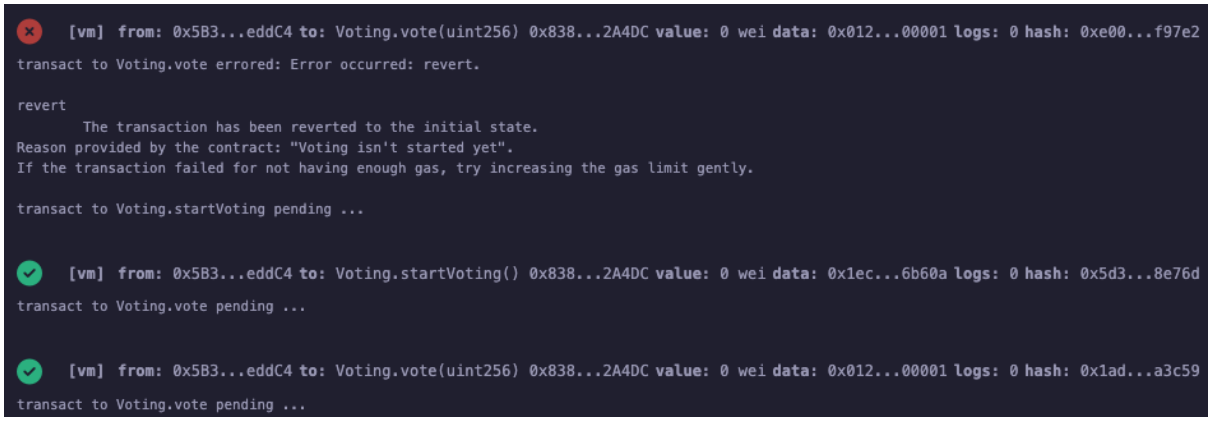
Даний метод є певною рулеткою, де користувач спалює в нікуди свої гроші, а натомість отримує шанс отримати додатковий голос.

4 Демо

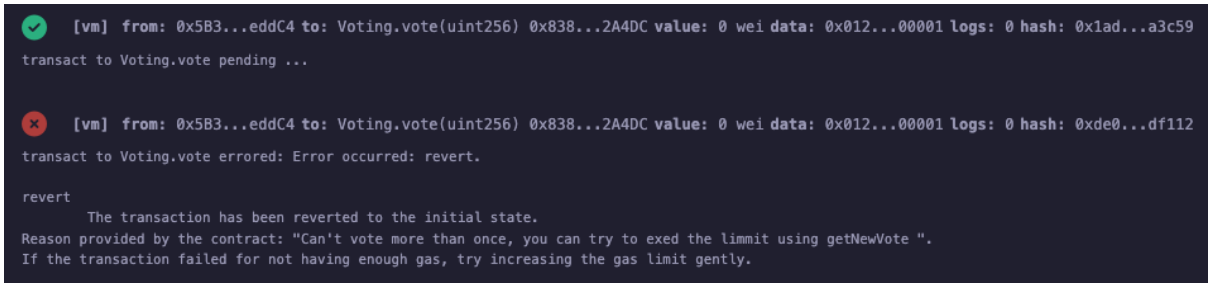
Наведемо приклад взаємодії з даним контрактом. В Remix IDE, доволі зручний інтерфейс. Можна однією кнопкою скомпілювати смартконтракт, після чого задеплоїти. Після цього з'явиться відповідна вкладка з наявними контрактами. Це виглядає наступним чином:



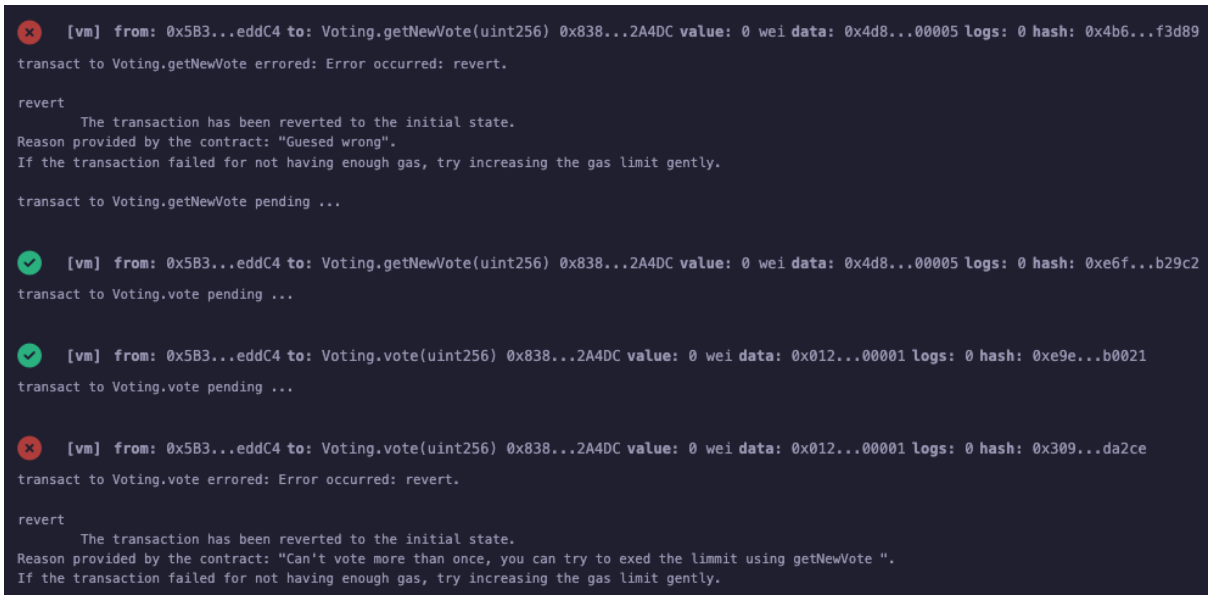
Проголосувати одразу, ми не можемо, адже адмін (той хто задеплоїв контракт) повинен почати голосування.



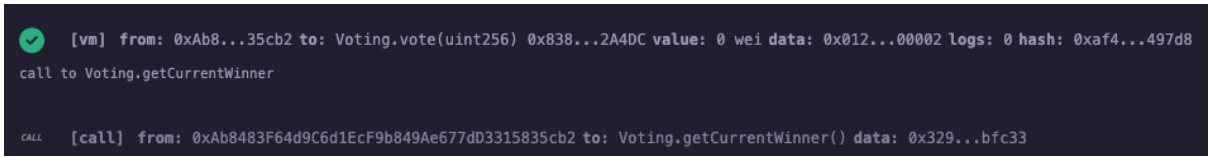
Кожний учасник може проголосувати лише один раз.

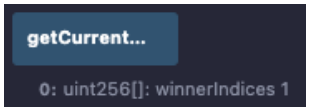


Але використовуючи механізм випадкової видачі нових голосів, можна голосувати допоки не закінчаться усі гроші.

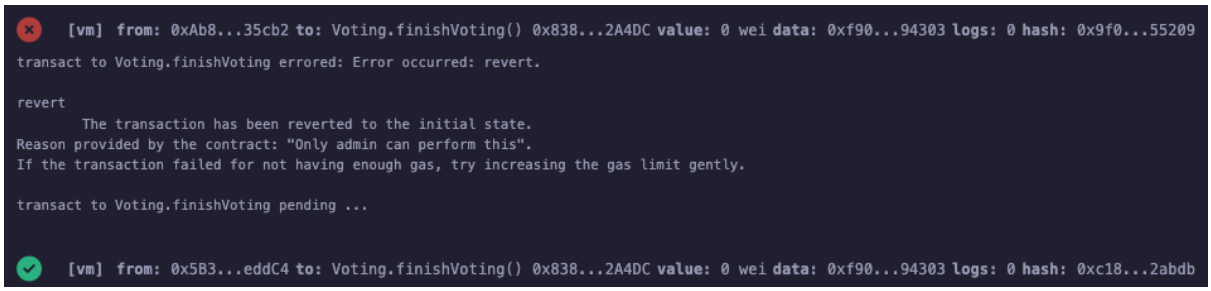


Також, ми можемо перейти на інші акаунти та проголосувати один раз з них. Викликавши функцію, що показує поточного виграшного кандидата, можемо побачити, що вигнає перший кандидат, адже за нього проголосовано два рази.





При спробі завершити голосуваня, видасться помилка, адже зараз ми на адмін. Перейшовши назад на акаунт адміна, бачимо, що операція успішна.



5 Висновки

У результаті виконання роботи було розроблено та протестовано смартконтракт системи голосування на мові Solidity у середовищі Remix IDE. Контракт реалізує базовий механізм виборів з адміністратором, що керує процесом, та довільними користувачами, яким надається право одного голосу. Для роботи механізму голосування було реалізовано допоміжний функціонал, наприклад: функціонал випадкового отримання нового голосу, функціонал керування станом голосування та інше. Робота дозволила ознайомитися з ключовими особливостями створення та взаємодії зі смартконтрактами.

Література

[God21] Hilda J Godlin. Create your first smart contract using Solidity in Remix IDE. https://medium.com/featurepreneur/create-your-first-smart-contract-using-solidity-in-remix-ide-354bb8a0eda0, 2021.

[sol] Solidity by Example. https://docs.soliditylang.org/en/v0.5.3/solidity-by-example.html.

A Програмний код

```
pragma solidity ^0.6.6;

contract Voting{

    struct Candidate{
        uint id;
        string name;
```

```
    uint voteCount;
}

address public admin;
mapping (uint => Candidate) public candidates;
uint public candidatecount;
mapping (address => bool) private voter;
bool public voting_started;
bool public voting_finished;

constructor() public{
    admin = msg.sender;
    voting_started = false;
    voting_finished = false;
    addCandidate("Godlin");
    addCandidate("Hilda");
}

modifier onlyAdmin() {
    require(msg.sender == admin, "Only admin can perform
        this");
    _;
}

modifier beeforeVoting() {
    require(!voting_started, "Voting not started yet");
    _;
}

function addCandidate(string memory _name) private{
    candidatecount++;
    candidates[candidatecount] = Candidate(candidatecount,
        _name, 0);
}

function vote(uint _candidateid) public{
    require(!voter[msg.sender], "Can't vote more than once,
        you can try to exed the limmit using getNewVote ");
    require(voting_started, "Voting isn't started yet");
    require(!voting_finished, "Voting is finished");

    voter[msg.sender] = true;
    candidates[_candidateid].voteCount ++;
}

function getNewVote(uint _rand_number) public {
```

```
        require(voter[msg.sender]);
        require(voting_started);
        require(!voting_finished, "Voting is finished");


        uint random = uint8(uint256(keccak256(abi.encodePacked(
            block.timestamp, block.difficulty))) % 10) + 1;
        require(random == _rand_number, "Guesed wrong");

        voter[msg.sender] = false;

    }

    function proposeCandidate(string memory _name) public{
        require(voting_started, "Voting not started yet");

        for (uint i = 0; i < candidatecount; i++) {
            if (keccak256(bytes(candidates[i].name)) ==
                keccak256(bytes(_name))) {
                revert("Candidate with this name already exists"
                    );
            }
        }

        addCandidate(_name);
    }

    function getCurrentWinner() public view returns (uint
        winnerIndex) {
        uint maxVotes = 0;
        for (uint i = 0; i <= candidatecount; i++) {
            if (candidates[i].voteCount > maxVotes) {
                maxVotes = candidates[i].voteCount;
                winnerIndex = i;
            }
        }
    }

    function startVoting() public onlyAdmin beeforeVoting {
        voting_started = true;
    }

    function finishVoting() public onlyAdmin {
        if (!voting_finished){
            voting_finished = true;
        }
```

```
    }

    function continueVoting() public onlyAdmin {
        require(voting_started);

        if (!voting_finished){
            voting_finished = true;
        }
    }
}
```