БЛОКЧЕЙН ТА ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ

Лабораторна робота №2

“Реалізація смарт-контракту або анонімної криптовалюти”

Недождій Максим, Буржимський Ростислав

ФІ-42мн

1 Мета роботи

Отримання навичок роботи із смарт-контрактами або анонімними криптовалютами.

Для другого типу лабораторних робіт

- розгортання та запуск обраної анонімної валюти, протоколювання майнінгу, пошук слідів деанонімізації;
- розгортання та запуск обраного смарт-контракту, підвищення ефективності роботи смарт-контракту з точки зору витрати гасу;
- розробка власного смарт-контракту.

ВАРІАНТ 4: ZCash.

2 Хід роботи

1. ZCash

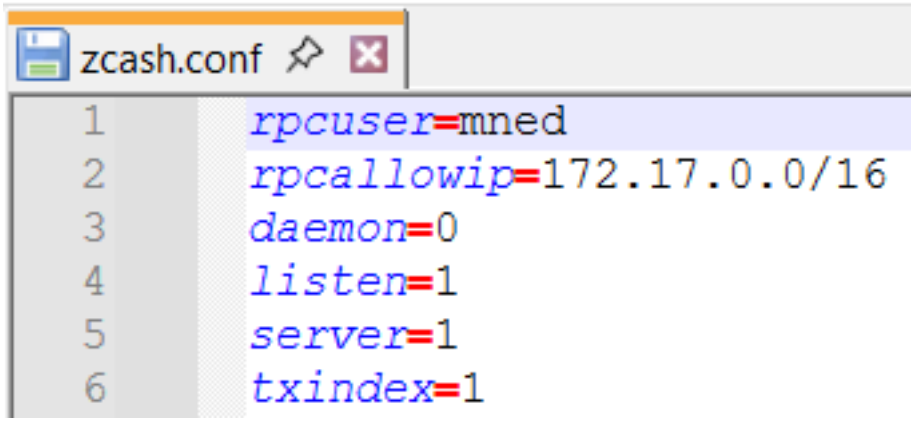
   Оскільки було обрано ZCash, необхідно встановити його на Docker.

```
PS C:\Users\Maks> docker pull zcashfr/zcash
Using default tag: latest
latest: Pulling from zcashfr/zcash
e62d08fa1eb1: Pull complete
f10e1bddd f3f: Pull complete
1cd62a315822: Pull complete
8ba2eca64c62: Pull complete
c1e9a37f19ec: Pull complete
9fe525454c71: Pull complete
39a92a2da2b5: Pull complete
2cae51d1db04: Pull complete
Digest: sha256:7f41072f70808e1457d8a3ca245d7ecef447ea8f90995f6028273269c065ff90
Status: Downloaded newer image for zcashfr/zcash:latest
docker.io/zcashfr/zcash:latest
```

   Далі необхідно створити два volume, які ми будемо використовувати для збереження даних

```
PS C:\Users\Maks> docker volume create zcash-data
zcash-data
PS C:\Users\Maks> docker volume create zcash-params
zcash-params
```

Створюємо zcash config, який необхідний для уточнення параметрів. Важливим полем є rpcallowip, оскільки 172.17.0.0/16 це стандартний IP, який використовує Docker.



Запускаємо контейнер zcash

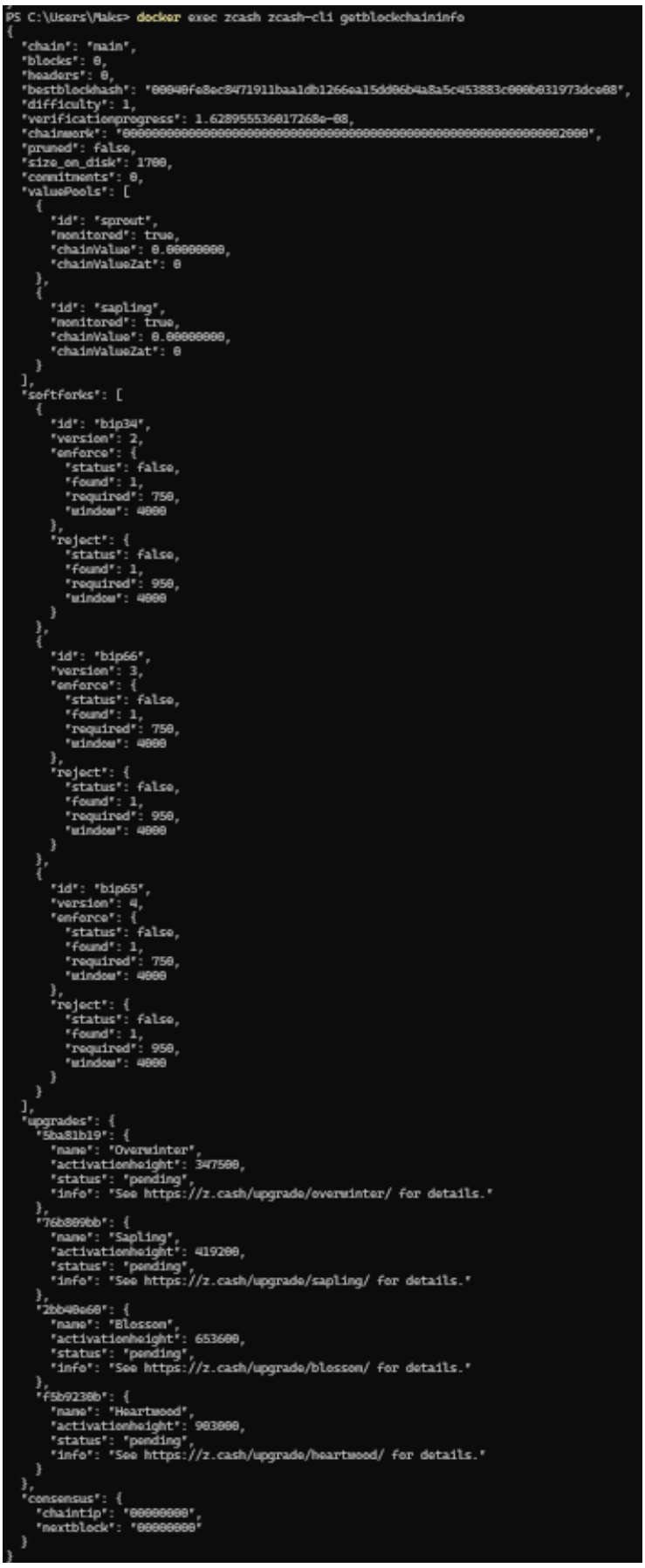
```
PS C:\Users\Maks> docker run -d --name zcash --user root -v zcash-data:/root/.zcash -v zcash-params:/root/.zcash-params
-v C:\Users\Maks\.docker\zcash.conf:/root/.zcash/zcash.conf -p 8232:8232 -p 127.0.0.1:8233:8333 zcashfr/zcash
3e5973084a987c9223a0ee5a289cf8ec9f5263d484f5aeed31e3ff262f724365
```

Перевіряємо чи успішно завершилась попередня команда

```
PS C:\Users\Maks> docker exec zcash zcash-cli getinfo
{
  "version": 3000050,
  "protocolversion": 170011,
  "walletversion": 60000,
  "balance": 0.00000000,
  "blocks": 0,
  "timeoffset": 0,
  "connections": 0,
  "proxy": "",
  "difficulty": 1,
  "testnet": false,
  "keypoololdest": 1748728980,
  "keypoolsize": 101,
  "paytxfee": 0.00000000,
  "relayfee": 0.00000100,
  "errors": ""
}
```

Бачимо, що помилок немає, але і синхронізація не активна

При перегляді розгорнутої інформації про блокчейн маємо наступний вигляд:



Можемо спробувати перевірити баланс, і побачимо, що у нашому розпоряджені порожній гаманець зі своїм сідом

```
PS C:\Users\Maks> docker exec zcash zcash-cli z_gettotalbalance
{
  "transparent": "0.00",
  "private": "0.00",
  "total": "0.00"
}
```

```
PS C:\Users\Maks> docker exec zcash zcash-cli getwalletinfo
{
  "walletversion": 60000,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 0,
  "keypoololdest": 1748728980,
  "keypoolsize": 101,
  "paytxfee": 0.00000000,
  "seedfp": "a984df7dc7660646715e87cdf388e43f6f90498a312d4fa9f814744a137fd147"
}
```

На відміну від лабораторної роботи №1, тут я не знайшов діючих faucet, з яких можна було б взяти валюту для тестування, але можна все одно спробувати створити адреси для майнінгу

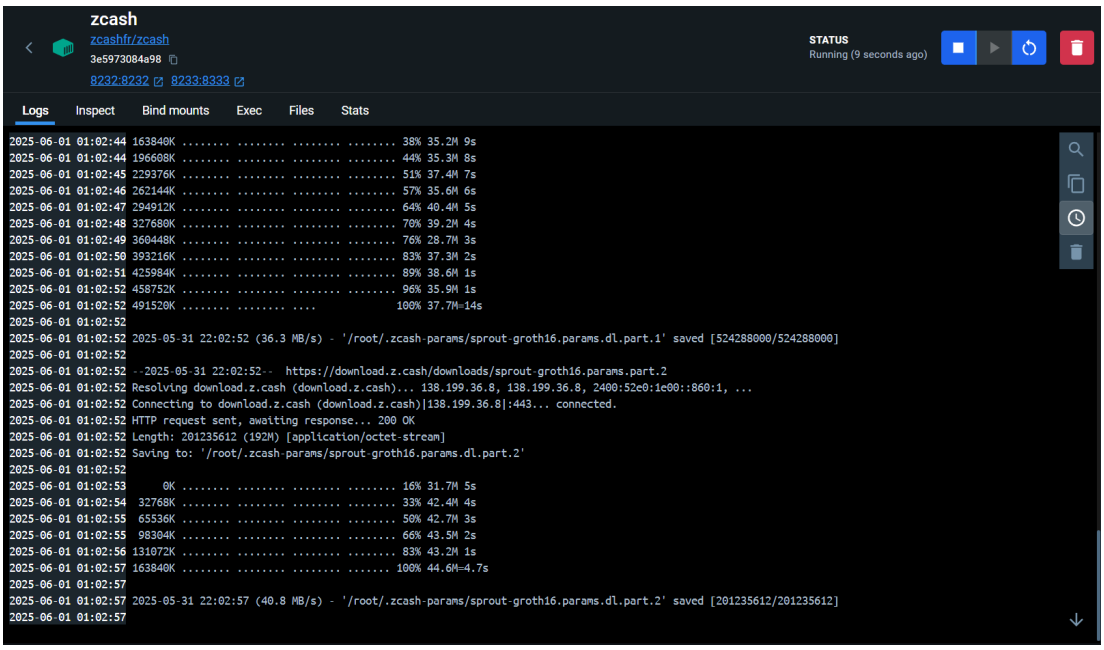
Створимо T- та Z-адреси

```
PS C:\Users\Maks> docker exec zcash zcash-cli getnewaddress
t1RJ32DBPeetQ9rzAUDo9xPfXGnrKu1U4Mc
PS C:\Users\Maks> docker exec zcash zcash-cli z_getnewaddress
zs1g4yh3wzny2tet7444dq03k2lk7v828wrfvwsrcnusxqv6xpu5xysp2hl0mumhp68hqzmqxsrr4d
```

Наступним кроком є запуск майнингу, який без повної синхронізації працювати не буде, лише імітувати роботу.

```
PS C:\Users\Maks> docker exec zcash zcash-cli getmininginfo
{
  "blocks": 0,
  "currentblocksize": 0,
  "currentblocktx": 0,
  "difficulty": 1,
  "errors": "",
  "genproclimit": 1,
  "localsolps": 0,
  "networksolps": 0,
  "networkhashps": 0,
  "pooledtx": 0,
  "testnet": false,
  "chain": "main",
  "generate": true
}
```

Протоколювання майнінгу можна було б вести по Logs у Docker Desktop container, або через zcashd.log



Пошук слідів деанонімізації можна виконати за рахунок мережевих утиліт, перевірки логів контейнера та аналізу взаємозв'язків між транзакціями з використанням прозорих T-адрес та екранованих Z-адрес, де нас цікавлять переходи коштів між адресами у цих двох групах ($Z \to T$ та $T \to Z$).

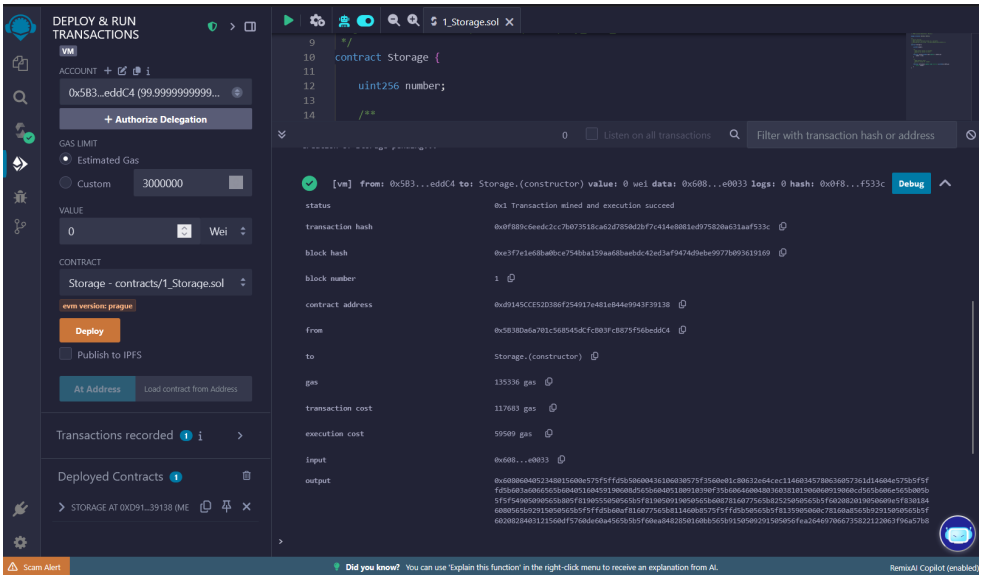
2. розгортання та запуск обраного смарт-контракту, підвищення ефективності роботи смарт-контракту з точки зору витрати гасу

   Для розгортання, запуску та підвищення ефективності смарт контракту достатньо використати RemixIDE, який дозволяє працювати зі смарт-контрактами онлайн.

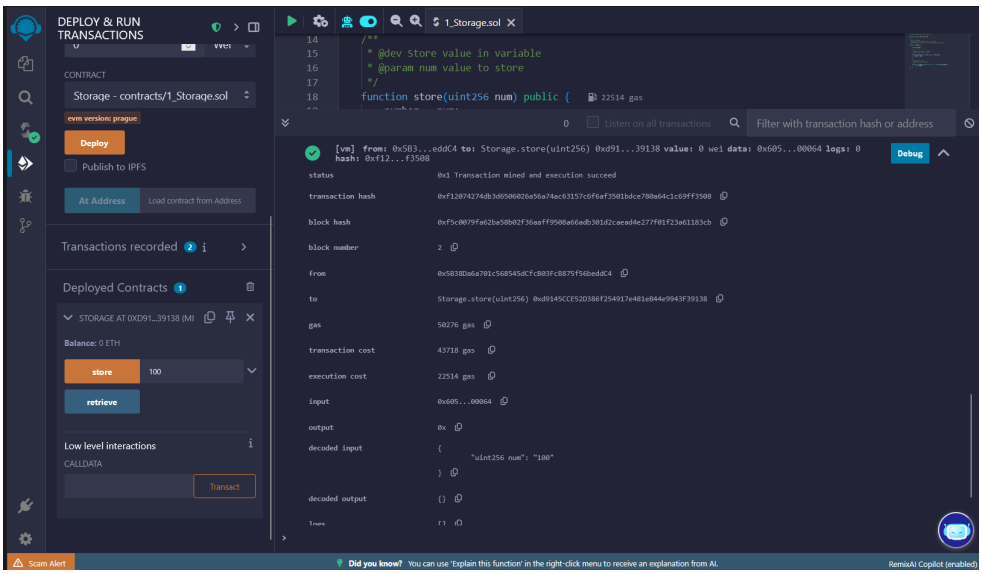
   Розглядатимемо найпростіший варіант контракту: Storage, який використовується для збереження даних. Він складається з двох частин: функція Store для вставлення даних у контейнер і функція Retrieve для повернення збереженого значення. Навіть у ньому можливо трохи покращити використання гасу.

   Витрачену кількість гасу можна побачити нижче.

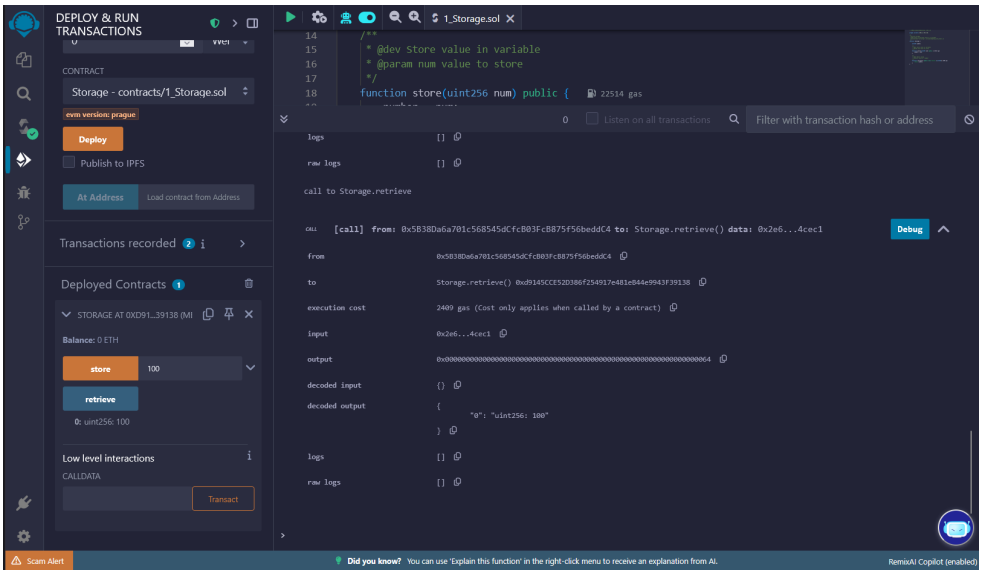
   Розгортання:



   Збереження:



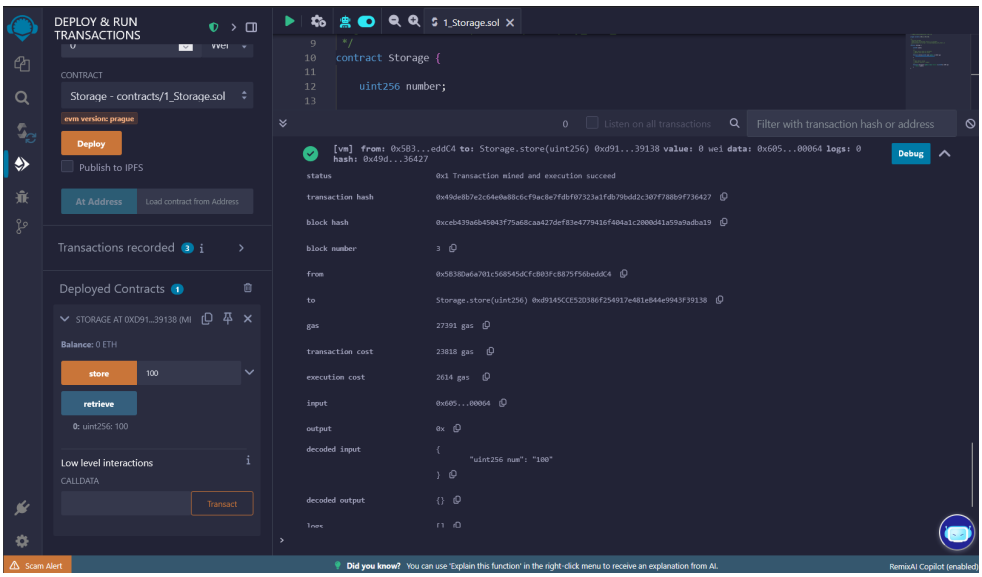
Відтворення:



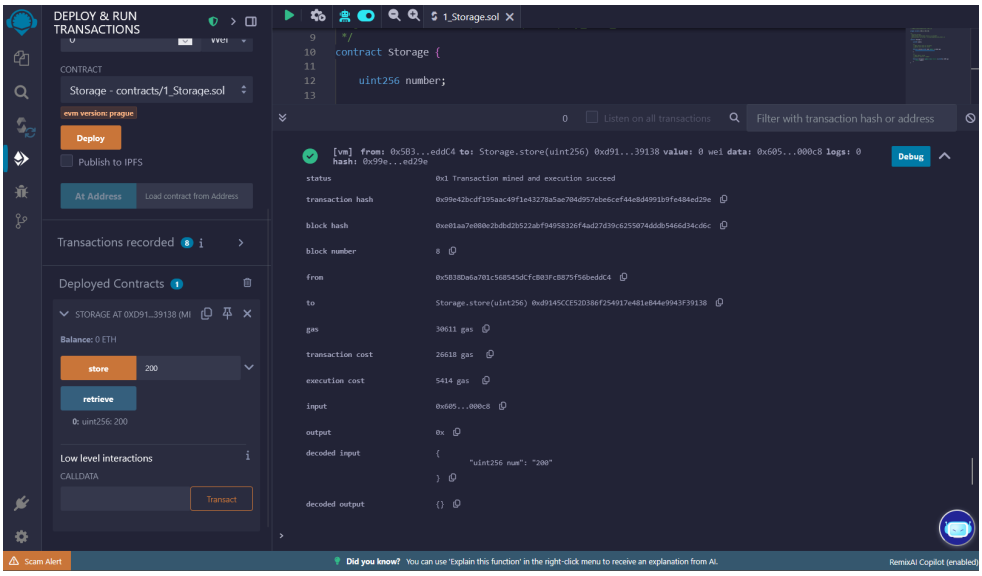
Оптимізувати процес збереження можна за рахунок вилучення зайвих операцій. Такими операціями є перезаписування значення, яке вже є у контейнері, ще раз у цей же контейнер.

```
if (number != num) number = num;
```

Таким чином, ми не витрачаємо гас на перезаписування значень, які вже зберігаються у контейнері.

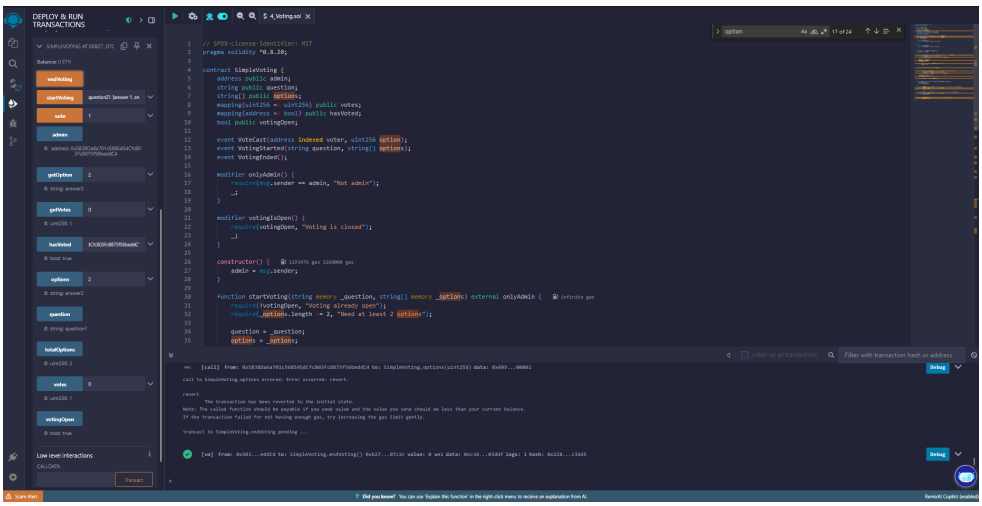


Бачимо, що записування іншого значення буде мати ту ж кількість гасу, але ми змогли зекономити на повторах, що має практичне застосування завдяки схильності у автоматизованих системах намагатись перезаписувати значення з певною періодичністю, що можна буде пропустити при повторі значення.



3. розробка власного смарт-контракту

Ми розробили смарт-контракт, який підтримує голосування. Одна адреса може проголосувати лише один раз, лише одне голосування може існувати одночасно. Для користування необхідно почати голосування надавши питання і масив відповідей, далі кожна адреса може проголосувати за варіант відповіді, лише адміністратор може почати або завершити голосування, будь-хто може переглянути питання, варіанти відповіді, скільки голосів було віддано за кожну відповідь, чи конкретна адреса давала відповідь, скільки всього варіантів відповіді існує, і чи йде зараз активне голосування.



Код:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract SimpleVoting {
    address public admin;
    string public question;
    string[] public options;
    mapping(uint256 => uint256) public votes;
    mapping(address => bool) public hasVoted;
    bool public votingOpen;

    event VoteCast(address indexed voter, uint256 option);
    event VotingStarted(string question, string[] options);
    event VotingEnded();

    modifier onlyAdmin() {
        require(msg.sender == admin, "Not admin");
        _;
    }

    modifier votingIsOpen() {
        require(votingOpen, "Voting is closed");
        _;
    }

    constructor() {
        admin = msg.sender;
    }

    function startVoting(string memory _question, string[] memory _options) external onlyAdmin {
        require(!votingOpen, "Voting already open");
        require(_options.length >= 2, "Need at least 2 options");

        question = _question;
```

```
        options = _options;
        votingOpen = true;

        for (uint256 i = 0; i < _options.length; i++) {
            votes[i] = 0;
        }

        emit VotingStarted(_question, _options);
    }

    function vote(uint256 optionIndex) external votingIsOpen {
        require(!hasVoted[msg.sender], "Already voted");
        require(optionIndex < options.length, "Invalid option");

        hasVoted[msg.sender] = true;
        votes[optionIndex] += 1;

        emit VoteCast(msg.sender, optionIndex);
    }

    function endVoting() external onlyAdmin votingIsOpen {
        votingOpen = false;
        emit VotingEnded();
    }

    function getOption(uint256 index) external view returns (string memory) {
        require(index < options.length, "Invalid option");
        return options[index];
    }

    function getVotes(uint256 index) external view returns (uint256) {
        require(index < options.length, "Invalid option");
        return votes[index];
    }

    function totalOptions() external view returns (uint256) {
        return options.length;
    }
}
```

3 Висновки

У ході лабораторної роботи було отримано практичний досвід роботи з вузлами криптовалюти, розгортанні та розробки смарт-контрактів. Досліджено можливості по скороченню використання витраченого гасу під час дій зі смарт-контрактами.