



Міністерство освіти і науки, молоді та спорту України  
Національний технічний університет України  
“Київський політехнічний інститут”  
Фізико-Технічний інститут

Технологія блокчейн та розподілені системи  
Лабораторна робота 2  
Реалізація смарт-контракту або анонімної криптовалюти

Перевірив:  
Селюх П.В.

---

Виконали:  
студенти групи  
ФБ-31мп  
Снігур Антон  
Тислицький Данііл  
Чорний Анатолій

Київ 2024

**Тема:** Реалізація смарт-контракту або анонімної криптовалюти.

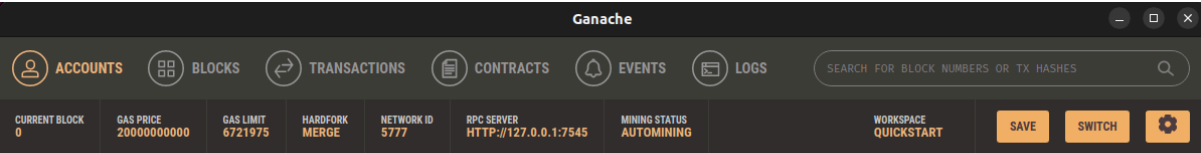
**Мета роботи:** «Отримання навичок роботи із смарт-контрактами або анонімними криптовалютами»

**Завдання на лабораторну роботу:**

Розробка власного смарт-контракту.

**Виконання завдання:**

Для виконання даної лабораторної роботи було вирішено спробувати реалізувати простий приклад власного смарт-контракту Ethereum. Для цього було розгорнуто локальний блокчейн Ethereum за допомогою графічної утиліти Ganache (<https://github.com/trufflesuite/ganache-ui>). Перевагою цього застосунку є те, що в один клік можна розгорнути тестову блокчейн мережу, що використовується для розробки смарт-контрактів. Вигляд Ganache UI показано на наступному скріншоті:

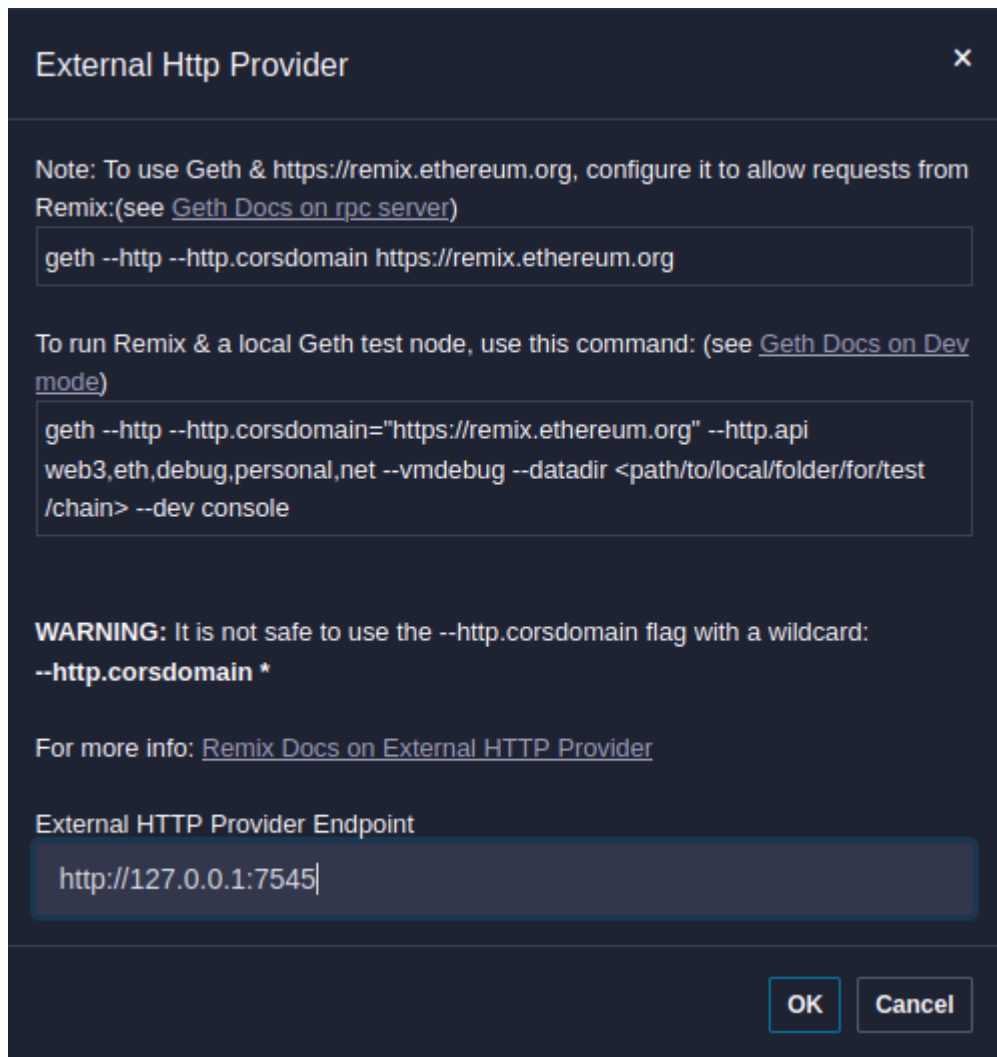


The screenshot shows the Ganache UI interface. At the top, there's a navigation bar with tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this is a status bar showing various network parameters like CURRENT BLOCK, GAS PRICE, GAS LIMIT, HARDFORK, NETWORK ID, RPC SERVER, and MINING STATUS. The main area displays a list of accounts. Each account row includes an ADDRESS, BALANCE, TX COUNT, and INDEX. The accounts are numbered 0 through 6, each with a unique hexadecimal address and a balance of 100.00 ETH. The TX COUNT is 0 for all accounts, and the INDEX is 0 through 6 respectively. There are also links to view the account details.

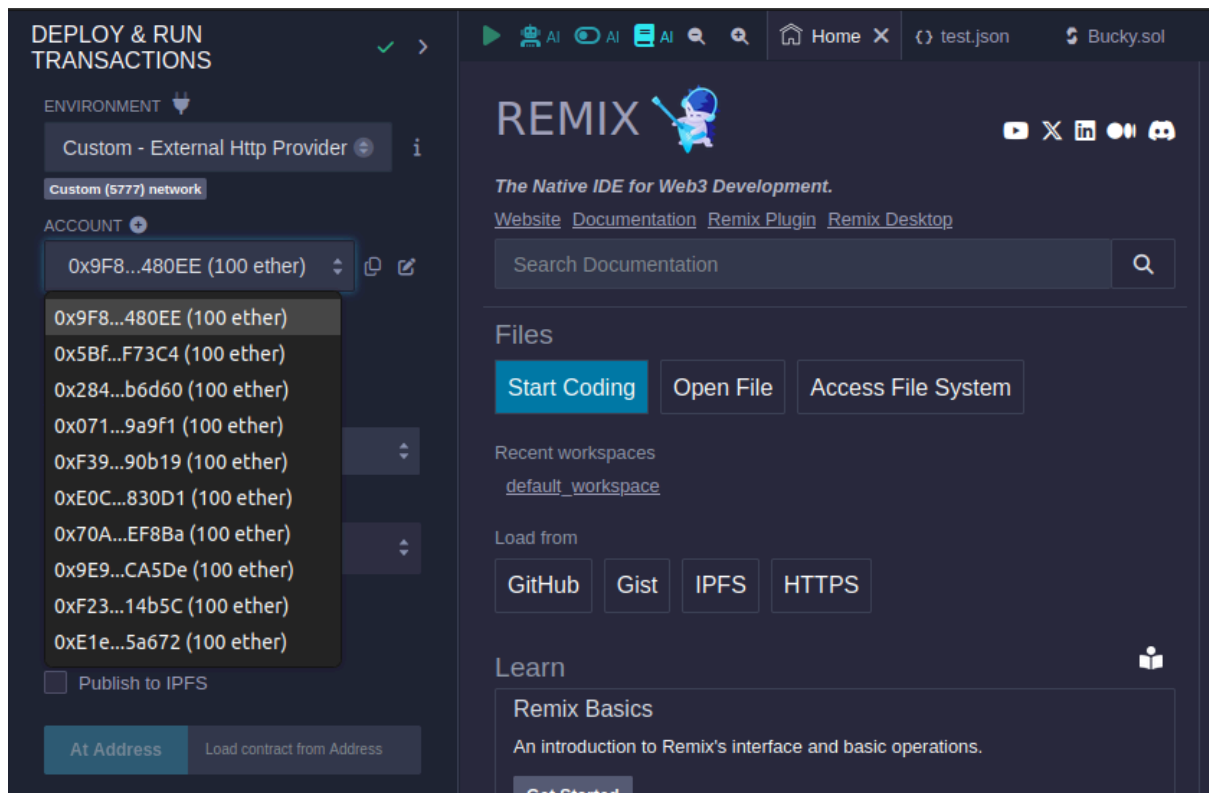
ADDRESS	BALANCE	TX COUNT	INDEX
0xFB34C3B66e62dB697336e703F88d6258125515A8	100.00 ETH	0	0
0xD3E30Ed81a2e778f5cb60c68d69768d6C01D6FBc	100.00 ETH	0	1
0xdc59B7931B02F6164f14B141B8453Af4A0F86f15	100.00 ETH	0	2
0x1fEfF8a6b800563db1f5feFc95Ff027b8D138C10	100.00 ETH	0	3
0x467597a112aC8EdE7EF0Fd4687F13Dc5D95f62E9	100.00 ETH	0	4
0xB92477126812486e6c2bB138C3Ef417Fff91b451	100.00 ETH	0	5
0xDF5e9FCf9aD199a228d3222f4d2B7b2c31541E0F	100.00 ETH	0	6

Для створення власного смарт-контракту необхідно використовувати спеціальний IDE для написання solidity коду. Для мережі Ethereum найбільш популярними IDE є Remix, Truffle, або Hardhat.

В даній роботі ми будемо використовувати Remix оскільки дана IDE реалізована у вигляді WEBView, а також дозволяє використовувати крім власних тестових блокчейн VM, Ethereum network розгорнуту у власній мережі, чим ми і скористаємося. Для цього необхідно у розділі **Deploy & run transactions** додати **Custom - External Http Provider Environment** та вказати http адресу нашого локального блокчейн:



Після цього ми будемо бачити ті самі тестові акаунти у Remix, що й показує Ganache UI:



Спробуємо реалізувати найпростіший solidity smart-contract, котрий буде виконувати операції setAge та getAge. Він буде виглядати наступним чином:

```

1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.8.7;
3
4  contract Bucky {
5      uint age;
6
7      function setAge(uint x) public { 22520 gas
8          age = x;
9      }
10
11     function getAge() public view returns (uint) { 2415 gas
12         return age;
13     }
14 }

```

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.7;
```

```
contract Bucky {
    uint age;

    function setAge(uint x) public {
        age = x;
    }
}
```

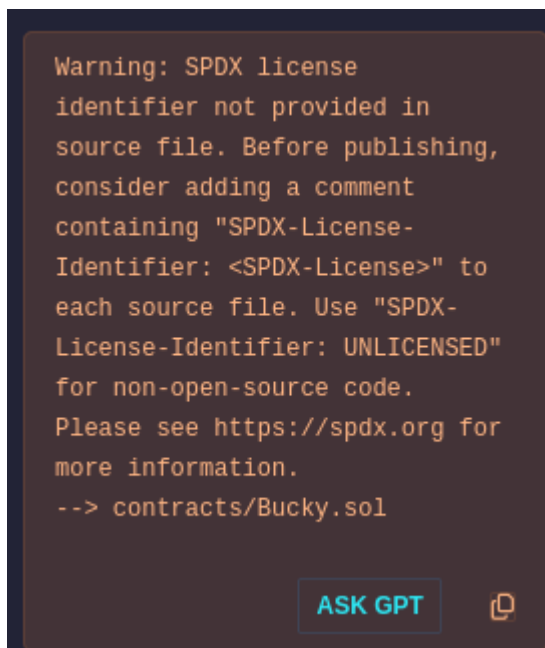
```

function getAge() public view returns (uint) {
    return age;
}
}

```

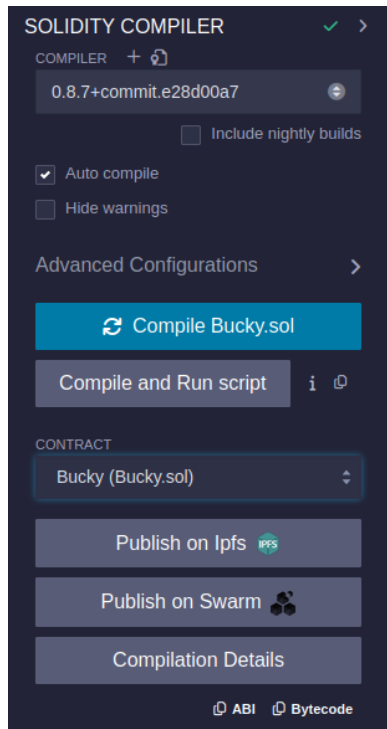
В даному простому смарт-контракті:

1. SPDX-License-Identifier: MIT: Цей рядок вказує ліцензію, за якою розповсюджується даний смарт-контракт. У цьому випадку використовується ліцензія MIT. (без вказування валідної ліцензії, процес компіляції повертає наступну помилку):

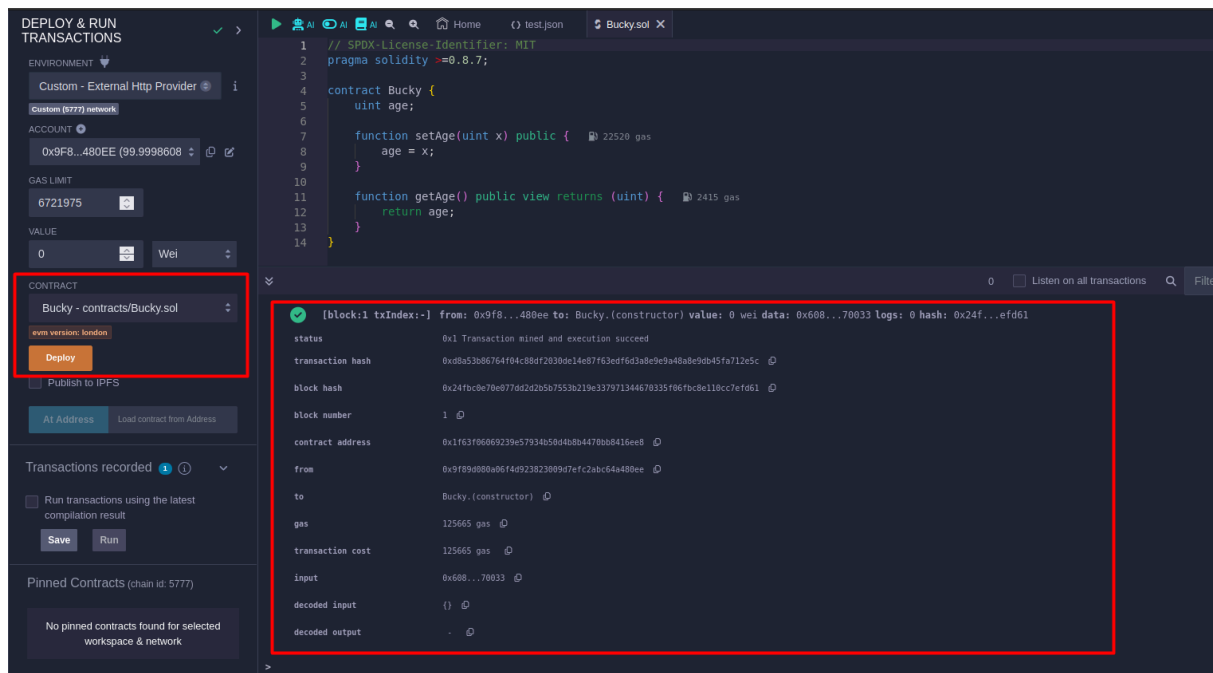


2. pragma solidity >=0.8.7;: Вказує версію компілятора Solidity, яка повинна бути використана для компіляції цього контракту. У даному випадку використана версія 0.8.7 або новіша.
3. contract Bucky: Контракт називається "Bucky".
4. uint age: Це змінна age, яка представляє вік. uint означає ціле число без знаку, що означає, що вона не може бути від'ємною.
5. function setAge(uint x) public: Це функція setAge, яка приймає вік x та присвоює його змінній age. Вона має звичайний модифікатор доступу public, що означає, що її можна викликати ззовні контракту.
6. function getAge() public view returns (uint): Це функція getAge, яка повертає вік. Має модифікатор доступу public та модифікатор view, що означає, що вона не змінює стан контракту та може бути викликана ззовні контракту.

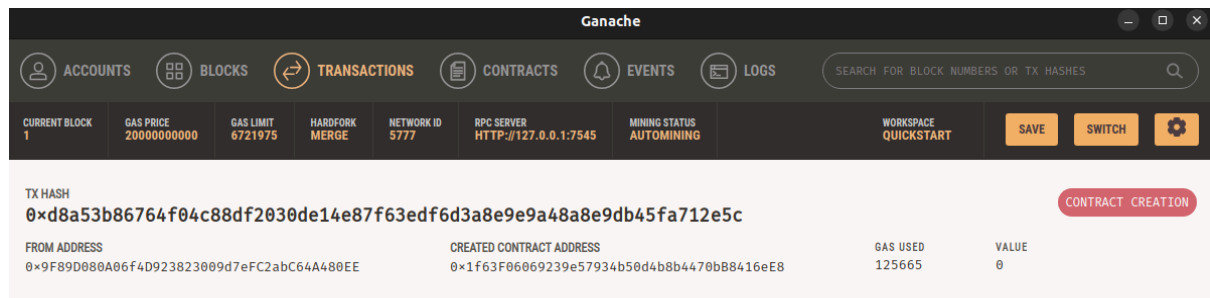
Після успішної компіляції отриманого смарт-контракту можемо побачити його к списку контрактів:



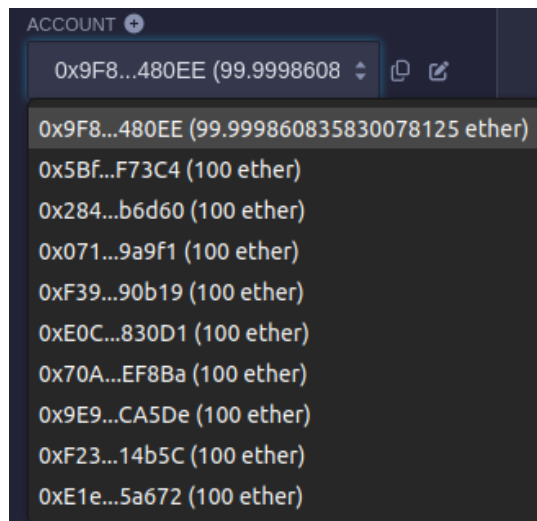
Для того, щоб розгорнути наш смарт-контракт на блокчейні необхідно виконати операцію deploy:



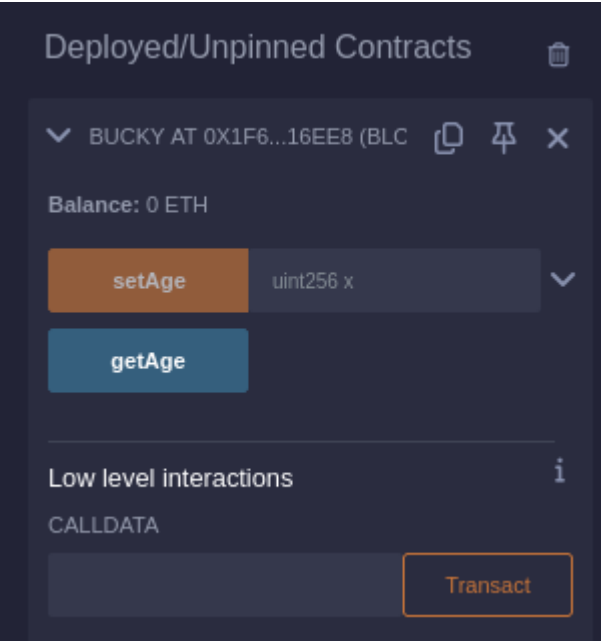
Як можемо побачити, операцію виконано успішно і вона використала 125665 GAS. Аналогічний output про створення контракту можемо побачити і у Ganache UI:



Також можемо побачити, що Account, котрий виконав деплой контракту має менше значення ETH:

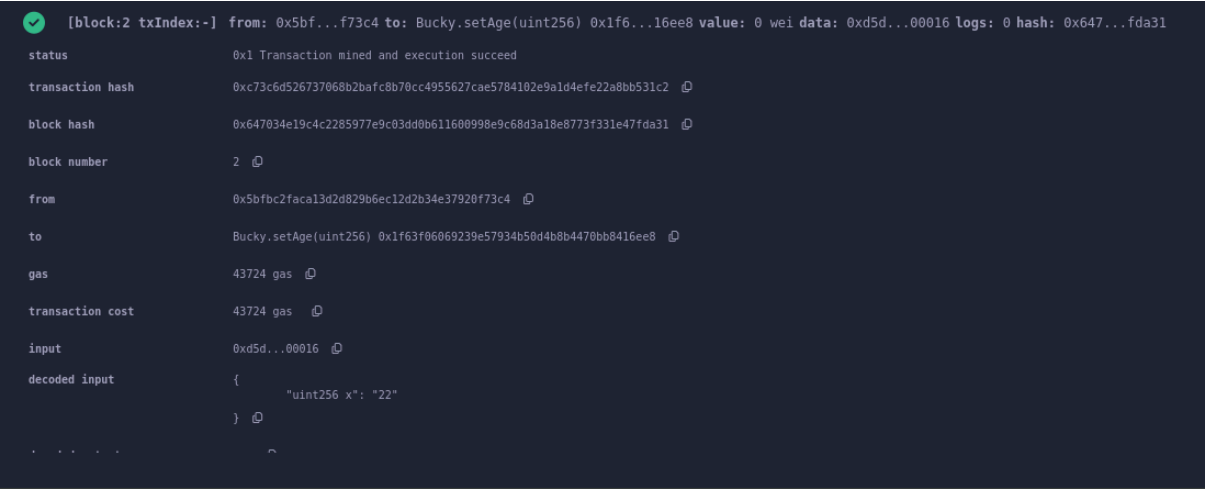


А смарт-контракт з'явився у полі **Deployed/Unpinned Contracts**:



Спробуємо виконати операції setAge getAge з різних акаунтів нашого блокчейн:

setAge:



TX HASH		CONTRACT CALL	
0xc73c6d526737068b2bafc8b70cc4955627cae5784102e9a1d4efe22a8bb531c2			
FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE
0x5BfBC2FacA13d2D829B6EC12D2b34e37920F73C4	0x1f63F06069239e57934b50d4b8b4470bb8416eE8	43724	0

getAge:



```
CALL [call] from: 0x28499cd52B6c25C33a29E3c86a941d428c5b6d60 to: Bucky.getAge() data: 0x967...e6e65

from      0x28499cd52B6c25C33a29E3c86a941d428c5b6d60
to        Bucky.getAge() 0x1f63F06069239e57934b50d4b8b4470bB8416eE8
input     0x967...e6e65
decoded input {}
decoded output {
    "0": "uint256: 22"
}
logs      []
```

Як можна побачити `getAge` операція не використовує `Gas`. Це відбувається через те, що газ використовується лише при виконанні змінних функцій, таких як `setAge()`, оскільки вони змінюють стан смарт-контракту.

Таким чином найпростіший смарт-контракт працює, спробуємо написати щось більш складне. Наприклад, власну реалізацію стандарту `ERC20`.

`ERC20` - це технічний стандарт, який визначає правила, які повинні відповідати всі токени, що базуються на блокчейні `Ethereum`, щоб їх можна було обмінювати між собою на основі розумних контрактів. `ERC20` був запропонований вперше в 2015 році. Основні функції `ERC20` включають передачу tokenів, отримання балансу tokenів та перевірку балансу користувачів.

Для цього необхідно реалізувати наступні операції:

- `TotalSupply`: Загальна кількість tokenів, які коли-небудь будуть випущені
- `BalanceOf`: Баланс рахунку власника tokenів
- `Transfer`: Автоматично виконує перекази вказаної кількості tokenів на вказану адресу для транзакцій з використанням токена
- `TransferFrom`: Автоматично виконує перекази вказаної кількості tokenів з вказаної адреси для транзакцій з використанням токена
- `Approve`: Дозволяє користувачеві зняти певну кількість tokenів з вказаного рахунку, до певної суми
- `Allowance`: Повертає певну кількість tokenів від користувача власнику

Для цього було реалізовано наступний смарт-контракт **MyToken.sol**:

```
// SPDX-License-Identifier: MIT
```

```

pragma solidity ^0.8.0;

contract MyToken {
    string public name;           // Назва токєну
    string public symbol;         // Символ токєну, наприклад, ETH
    uint8 public decimals;        // Кількість десяткових знаків після коми
    uint256 public totalSupply;   // Загальна кількість токєнів, яка була
    створєна

    // Зберігає баланс кожного власника токєнів
    mapping (address => uint256) public balanceOf;

    // Зберігає дозволи на витрачання токєнів
    mapping (address => mapping (address => uint256)) public allowance;

    // Подія, яка відбувається при кожному успішному переказі токєнів
    event Transfer(address indexed from, address indexed to, uint256 value);

    // Подія, яка відбувається при кожному успішному наданні дозволу на витрачання
    токєнів
    event Approval(address indexed owner, address indexed spender, uint256 value);

    // Конструктор контракту, який встановлює загальний обсяг випуску, назву, символ
    і кількість десяткових знаків
    constructor(
        uint256 initialSupply,
        string memory tokenName,
        string memory tokenSymbol,
        uint8 decimalUnits
    ) {
        totalSupply = initialSupply * 10 ** uint256(decimalUnits); // Встановлення
        загального обсягу випуску
        balanceOf[msg.sender] = totalSupply; // Встановлення балансу власника
        контракту (тобто того, хто викликав конструктор)
        name = tokenName; // Встановлення назви токєну
        symbol = tokenSymbol; // Встановлення символу токєну
        decimals = decimalUnits; // Встановлення кількості десяткових знаків після
        коми
    }

    // Внутрішня функція для здійснення переказу токєнів з одного рахунку на інший
    function _transfer(address _from, address _to, uint _value) internal {
        require(_to != address(0)); // Перевірка на дійсність адреси отримувача
        require(balanceOf[_from] >= _value); // Перевірка чи є власника достатньо
        токєнів для переказу
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Перевірка на
        переповнення
        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        balanceOf[_from] -= _value; // Віднімання токєнів з рахунку відправника
        balanceOf[_to] += _value; // Додавання токєнів на рахунок отримувача
        emit Transfer(_from, _to, _value); // Генерація події переказу токєнів
    }
}

```

```

        assert(balanceOf[_from] + balanceOf[_to] == previousBalances); //
Перевірка чи баланси після переказу є коректними
    }

    // Функція для переказу токенів з одного рахунку на інший
    function transfer(address _to, uint256 _value) public returns (bool success) {
        _transfer(msg.sender, _to, _value); // Виклик внутрішньої функції _transfer
для здійснення переказу
        return true;
    }

    // Функція для переказу токенів з одного рахунку на інший через проміжний рахунок
    function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success) {
        require(_value <= allowance[_from][msg.sender]); // Перевірка дозволу на
витрату токенів
        allowance[_from][msg.sender] -= _value; // Зменшення дозволу на витрату
токенів
        _transfer(_from, _to, _value); // Виклик внутрішньої функції _transfer для
здійснення переказу
        return true;
    }

    // Функція, що дає дозвіл іншому користувачу на витрату певної кількості токенів з
вашого рахунку
    function approve(address _spender, uint256 _value) public returns (bool
success) {
        allowance[msg.sender][_spender] = _value; // Встановлення дозволу на
витрату токенів
        emit Approval(msg.sender, _spender, _value); // Генерація події надання
дозволу на витрату токенів
        return true;
    }
}

```

Приклад виконання:

Деплой контракту токена КРІ з символом ІРТ у кількості 30000 екземплярів (виклик конструктору):

**DEPLOY**

INITIALSUPPLY:

30000

TOKENNAME:


KPI


TOKENSYMBOL:

IPT


DECIMALUNITS:

1

 Calldata

 Parameters


transact

 [vm] from: 0x5B3...eddC4 to: MyToken.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0x11b...2f25f


status

0x1 Transaction mined and execution succeed


transaction hash

0x11bd4d43b04ec4f8f32d6d4d1791e1aaae61d2e7d8a91f03471b71dc7772f25f 


block hash

0x4e1432fae4da9c7aba589b33e34a81b807273a3bf4bba5c327f3e1d885184d59 


block number

21 


contract address

0xddaAd340b0f1Ef65169Ae5E41A8b10776a75482d 


from

0x5B38Da6a701c568545dCfc803Fc8875f56beddC4 

to

MyToken.(constructor) 

gas

1042316 gas 

Перевірка констант, що задані при деплоїменті:

decimals

0: uint8: 1

name

0: string: KPI

symbol

0: string: IPT

totalSupply

0: uint256: 300000

```

call to MyToken.totalSupply

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MyToken.totalSupply() data: 0x181...60ddd
call to MyToken.symbol

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MyToken.symbol() data: 0x95d...89b41
call to MyToken.name

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MyToken.name() data: 0x06f...dde03
call to MyToken.decimals

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: MyToken.decimals() data: 0x313...ce567

```

Перевірка балансу акаунту, що задеплоїв смарт-контракт (0x5B38Da6a701c568545dCfcB03FcB875f56beddC4):

Перевід токену на інший акаунт з поточного:

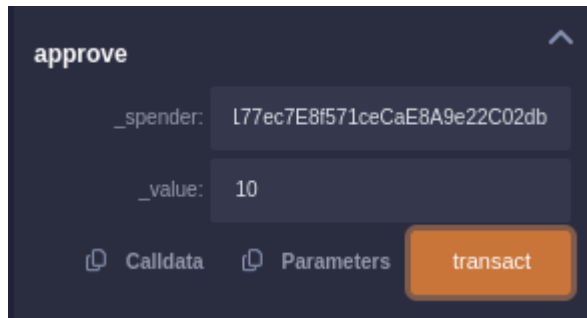
Баланс акаунту 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2:

Якщо на акаунті недостатньо токенив, отримуємо помилку:

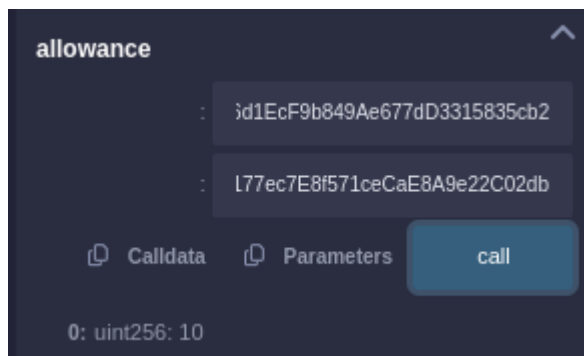
```
✖ [vm] from: 0xAb8...35cb2 to: MyToken.transfer(address,uint256) 0xdda...5482d value: 0 wei data: 0xa90...003e8 logs: 0 hash: 0x3b1...58b99
transact to MyToken.transfer errored: Error occurred: revert.

revert
    The transaction has been reverted to the initial state.
Note: The called function should be payable if you send value and the value you send should be less than your current balance.
Debug the transaction to get more information.
```

Надавання дозволу акаунту **0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db** переводити 10 IPT токенів з акаунту **0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2**:

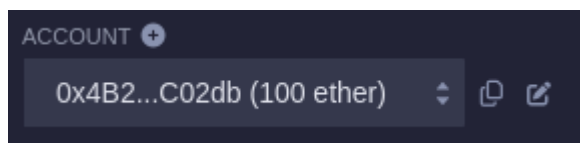


Перевірка, що це дозволено за допомогою операції Allowance:



Перевірка операції TransferFrom:

Знаходимося на акаунті **0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db**:



Спроба виконати перевід 15 токенів з акаунту **0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2** на акаунт **0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB**:

transferFrom

\_from:

0d1EcF9b849Ae677dD3315835cb2

\_to:

034aC0F824c42a7cC18A495cabaB

\_value:

15

Calldata

Parameters

transact

Отримуємо помилку, оскільки дозволено переводити лише 10 токенів.  
Спробуємо аналогічно з 5 токенами:

transferFrom

\_from:

0d1EcF9b849Ae677dD3315835cb2

\_to:

034aC0F824c42a7cC18A495cabaB

\_value:

5

Calldata

Parameters

transact

Операцію виконано успішно:

✓ [vm]	from: 0x4B2...C02db to: MyToken.transferFrom(address,address,uint256) 0xdda...5482d value: 0 wei data: 0x23b...00005 logs: 1 hash: 0x83b...703bf
status	0x1 Transaction mined and execution succeed
transaction hash	0x83b4abf7fab0641f37ce4434155182bf2f7c3288e4125a4dd3ffa74fb0b703bf
block hash	0x6c415752cfdea501f3ef2590512de7fd221507497c52925d9481e54941754c80
block number	26
from	0x4B209938c481177ec7E8f571ceCaE8A9e22C02db
to	MyToken.transferFrom(address,address,uint256) 0xddaAd340b0f1Ef65169Ae5E41A8b10776a75482d
gas	69467 gas
transaction cost	60406 gas
execution cost	38466 gas

Використані джерела:

<https://medium.com/coinmonks/solidity-tutorial-how-to-use-remix-ide-for-solidity-smart-contract-development-d0d2ce6da051>

<https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-setup-local-development-environment-for-solidity/how-to-setup-local-development-environment-for-solidity>

<https://medium.com/@adamh90/creating-a-local-test-environment-for-ethereum-smart-contracts-1f638efca020>

<https://solidity-by-example.org/app/erc20/>

<https://github.com/trufflesuite/ganache-ui>

<https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/>