# Department of Computer Engineering

Academic Term: Jan-May 23-24

**Class:** B.E Computer Sem -VII **Subject:** Blockchain Technology Lab **Subject Code :** CSDL7022

|  |  |
| --- | --- |
| **Practical No:** | **5** |
| **Title:** | **Embedding wallet and transaction using Solidity** |
| **Date of Performance:** | **25/08/2023** |
| **Date of Submission:** | **25/08/2023** |
| **Roll No:** | **9427** |
| **Name of the Student:** | **Atharva Prashant Pawar** |

Evaluation:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **Time Line (2)** |  |
| **2** | **Output (3)** |  |
| **3** | **Code optimization (2)** |  |
| **4** | **Post lab (3)** |  |

**Signature of the Teacher :**

# Experiment No. 5

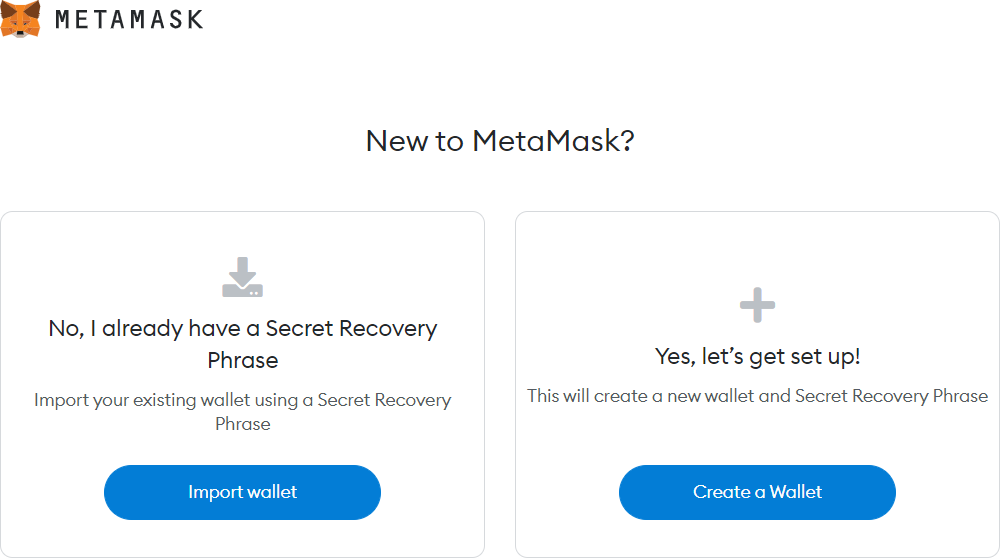
**Embedding wallet and transaction using Solidity**

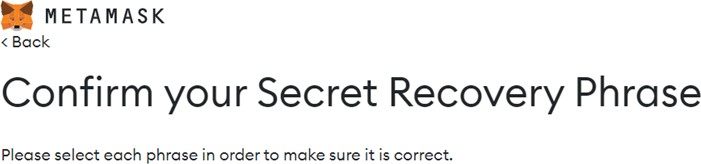
**Aim:** Creating a smart contract using Metamask and performing the transactions.

# Theory:

## Step 1: Create a wallet at meta-mask

* Install MetaMask in Chrome browser and enable it.



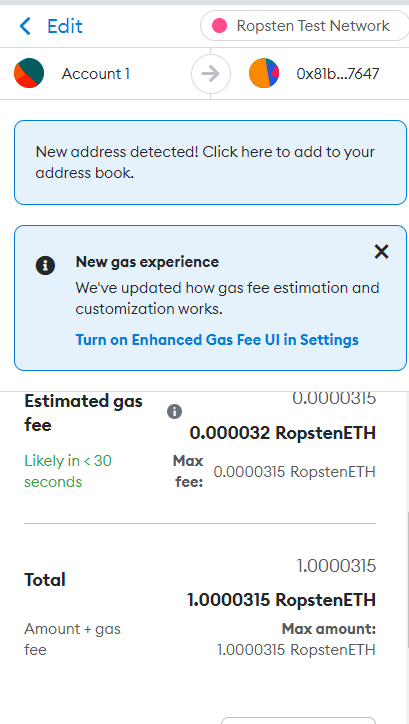


## Step 2: Select any one test network

Click on show networks. Enable show networks and other information. Following test networks can be seen in your MetaMask wallet. These networks are only for the purpose of testing and ethers involved in it have no value.

* Ropsten Test Network
* Kovan Test Network
* Rinkeby Test Network
* Goerli Test Network

Select Ropsten Test Network.

**Step 3:** Add some dummy Ethers to your wallet

* To test the smart contract, your MetaMask wallet should contain some dummy ethers. For example, if you test a contract using the Ropsten test network, select it and you will find 0 ETH as the initial balance in your account. Click on Buy button. Buy test ethers.

**Step 4:** Compile and Deploy following contract in Ethereum Remix IDE. Give name to file token.sol. Deploy by selecting **QKCToken–contract/token.sol**. **Select Environment as Injected Provider-Metamask.** Replace **YOUR\_METAMASK\_WALLET\_ADDRESS** in code by your wallet address which you have created.

pragma solidity ^0.8.08;

//Safe Math Interface contract SafeMath {

function safeAdd(uint a, uint b) public pure returns (uint c) { c = a + b;

require(c >= a);

}

function safeSub(uint a, uint b) public pure returns (uint c) { require(b <= a);

c = a - b;

}

function safeMul(uint a, uint b) public pure returns (uint c) { c = a \* b;

require(a == 0 || c / a == b);

}

function safeDiv(uint a, uint b) public pure returns (uint c) { require(b > 0);

c = a / b;

}

}

//ERC Token Standard #20 Interface contract ERC20Interface {

function totalSupply() public constant returns (uint);

function balanceOf(address tokenOwner) public constant returns (uint balance);

function allowance(address tokenOwner, address spender) public constant returns (uint remaining);

function transfer(address to, uint tokens) public returns (bool success); function approve(address spender, uint tokens) public returns (bool success);

function transferFrom(address from, address to, uint tokens) public returns (bool success);

event Transfer(address indexed from, address indexed to, uint tokens);

event Approval(address indexed tokenOwner, address indexed spender, uint tokens);

}

//Contract function to receive approval and execute function in one call contract ApproveAndCallFallBack {

function receiveApproval(address from, uint256 tokens, address token, bytes data) public;

}

//Actual token contract

contract QKCToken is ERC20Interface, SafeMath { string public symbol;

string public name; uint8 public decimals; uint public \_totalSupply;

mapping(address => uint) balances;

mapping(address => mapping(address => uint)) allowed;

constructor() public { symbol = "QKC";

name = "QuikNode Coin"; decimals = 2;

\_totalSupply = 100000; balances[YOUR\_METAMASK\_WALLET\_ADDRESS] = \_totalSupply;

emit Transfer(address(0), YOUR\_METAMASK\_WALLET\_ADDRESS, \_totalSupply);

}

function totalSupply() public constant returns (uint) { return \_totalSupply - balances[address(0)];

}

function balanceOf(address tokenOwner) public constant returns (uint balance) { return balances[tokenOwner];

}

function transfer(address to, uint tokens) public returns (bool success) { balances[msg.sender] = safeSub(balances[msg.sender], tokens); balances[to] = safeAdd(balances[to], tokens);

emit Transfer(msg.sender, to, tokens); return true;

}

function approve(address spender, uint tokens) public returns (bool success) { allowed[msg.sender][spender] = tokens;

emit Approval(msg.sender, spender, tokens); return true;

}

function transferFrom(address from, address to, uint tokens) public returns (bool success) { balances[from] = safeSub(balances[from], tokens);

allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens); balances[to] = safeAdd(balances[to], tokens);

emit Transfer(from, to, tokens);

return true;

}

function allowance(address tokenOwner, address spender) public constant returns (uint remaining) {

return allowed[tokenOwner][spender];

}

function approveAndCall(address spender, uint tokens, bytes data) public returns (bool success) {

allowed[msg.sender][spender] = tokens; emit Approval(msg.sender, spender, tokens);

ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data); return true;

}

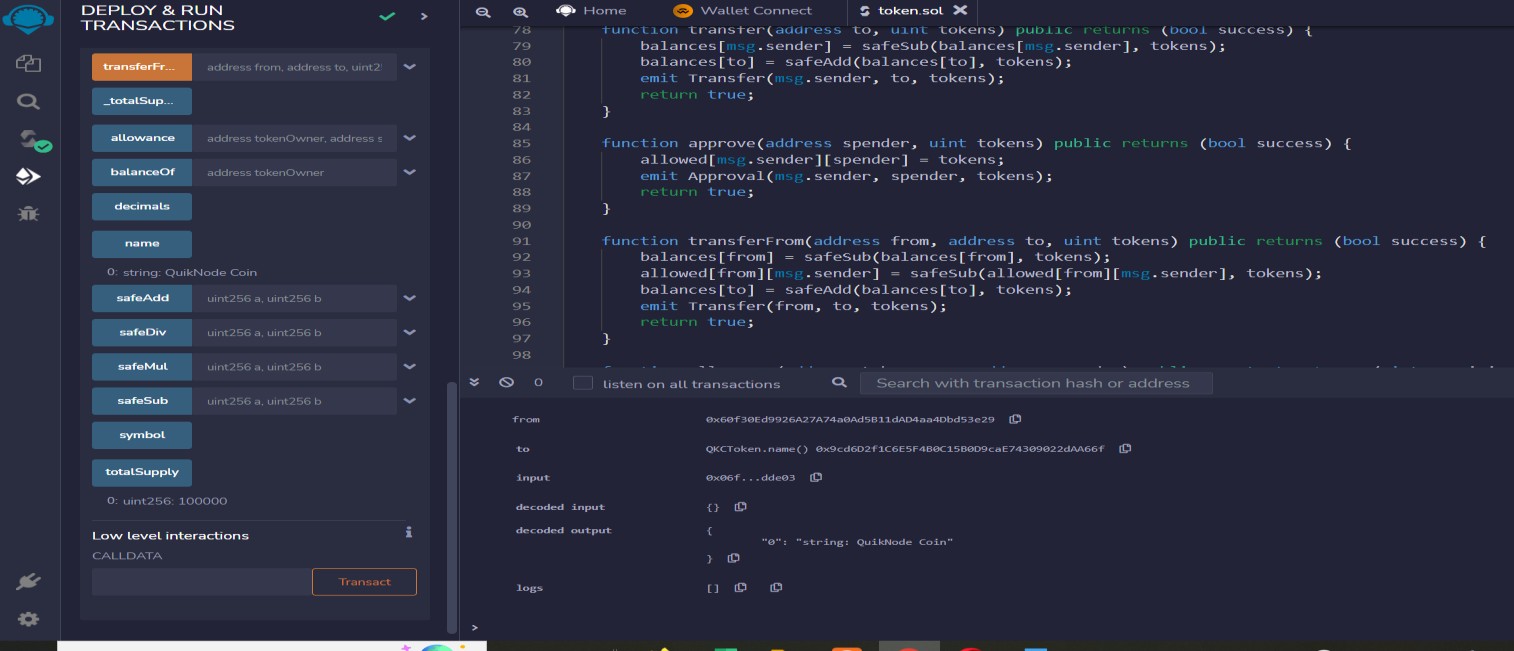
function () public payable { revert();

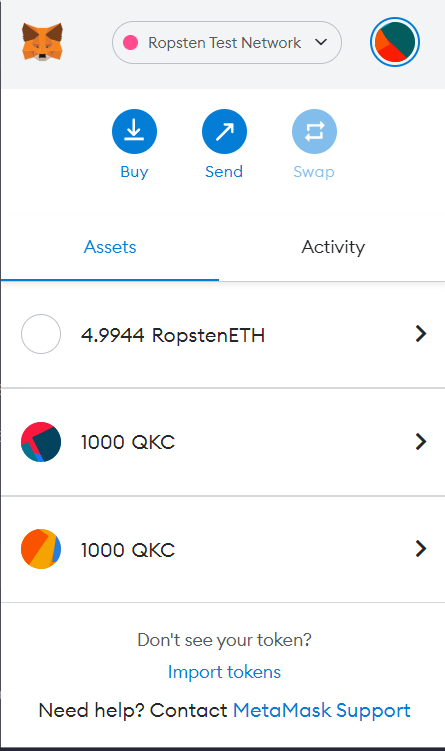
}

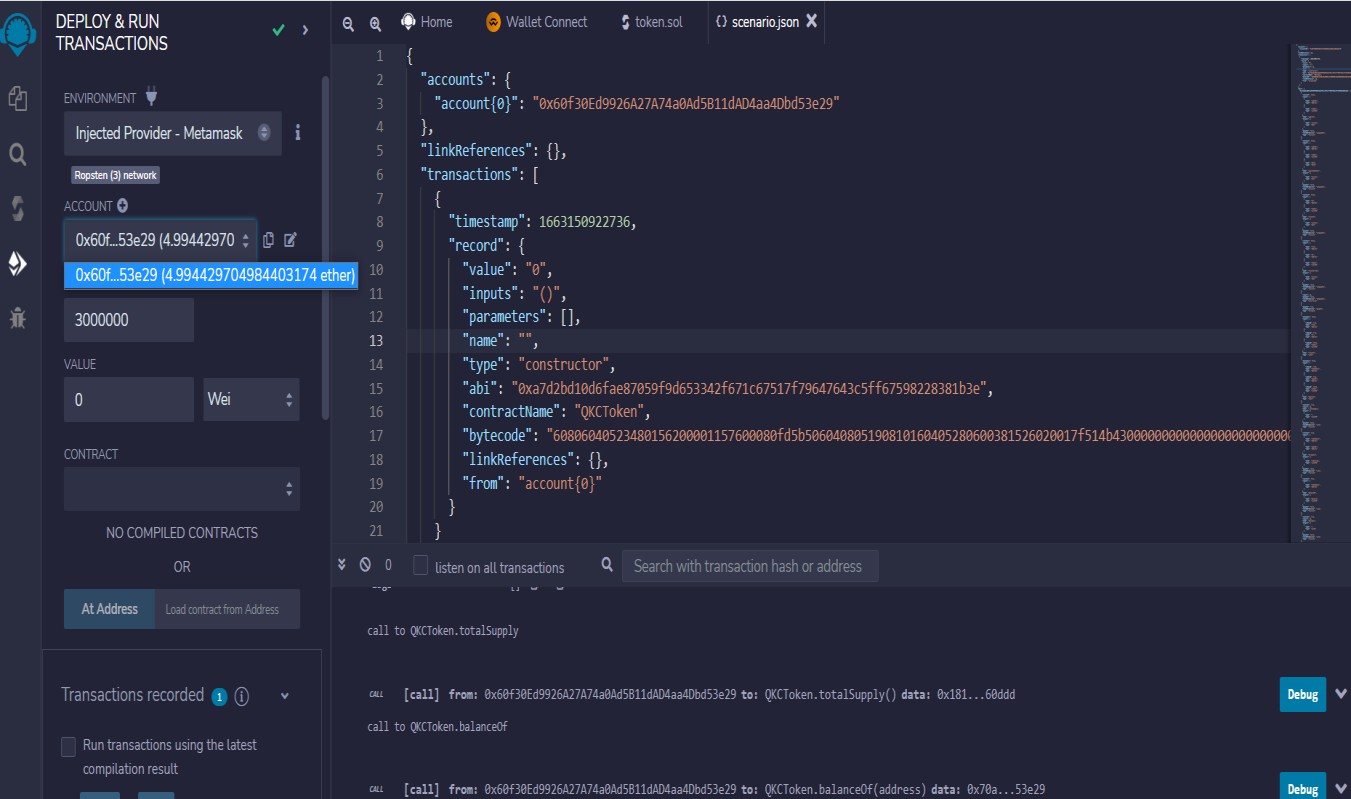
}

## Methods

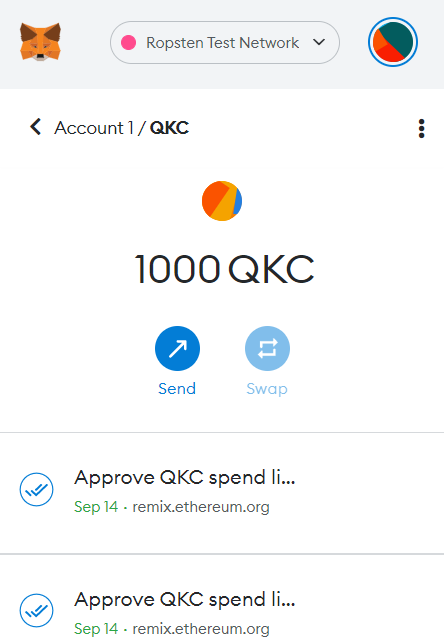
**Step 5: totalSupply**: A method that defines the total supply of your tokens, When this limit is reached the smart contract will refuse to create new tokens. As defined in contract total supply is 100000 shown below.



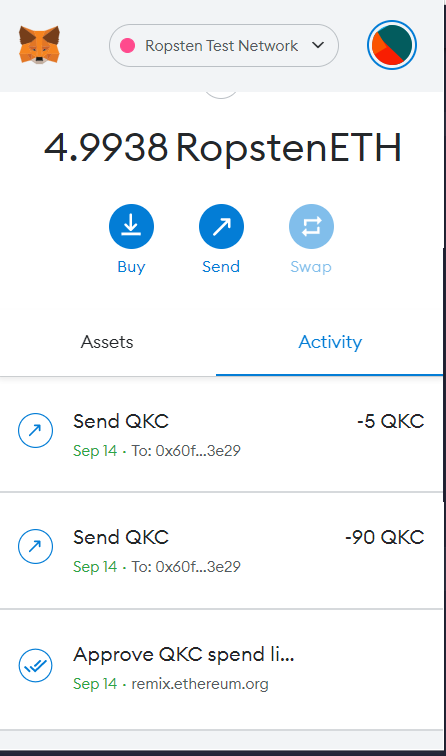




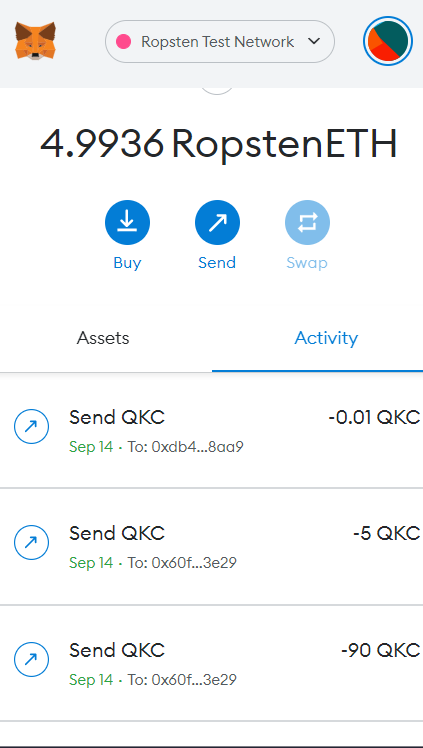
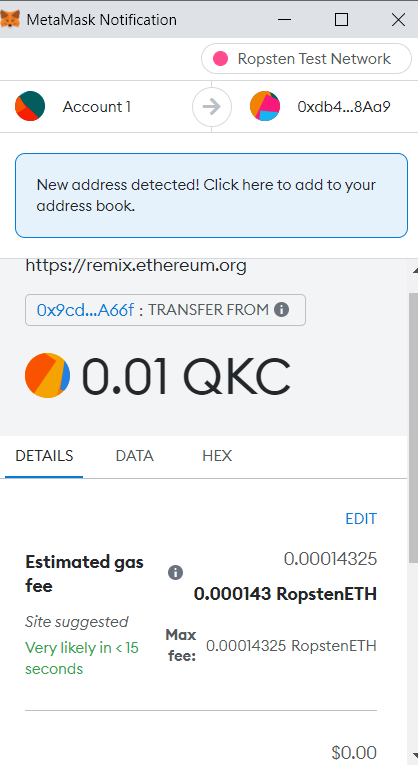
**balanceOf**: A method that returns the number of tokens a wallet address has.



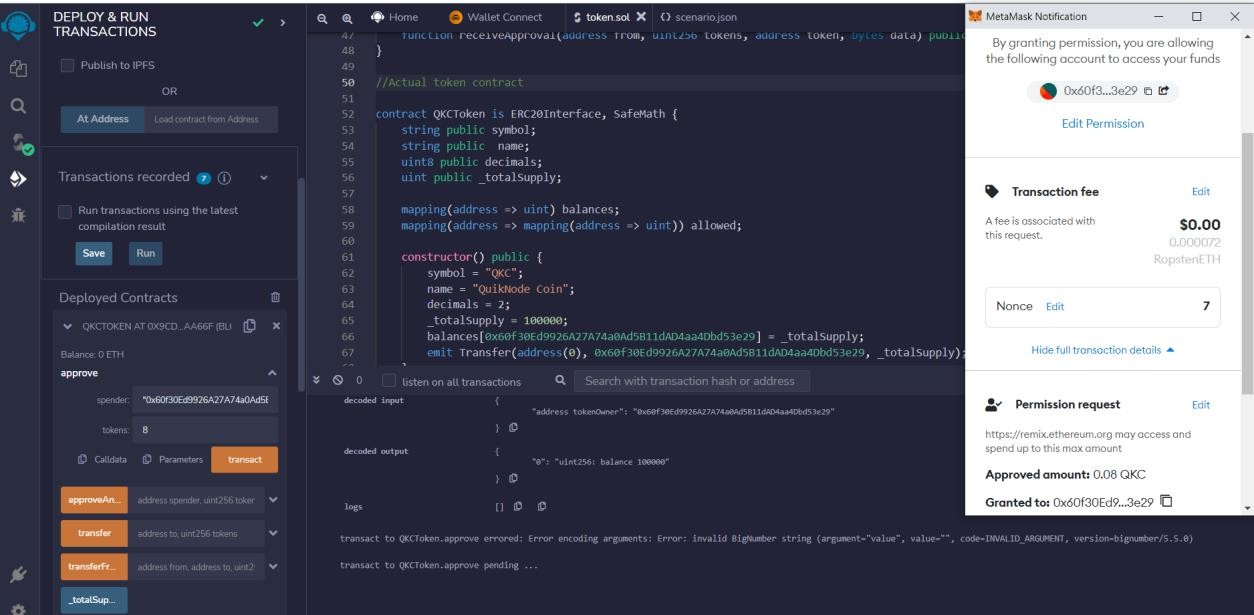
**transfer**: A method that takes a certain amount of tokens from the total supply and gives it to a user.



**transferFrom**: Another type of transfer method which is used to transfer tokens between users.



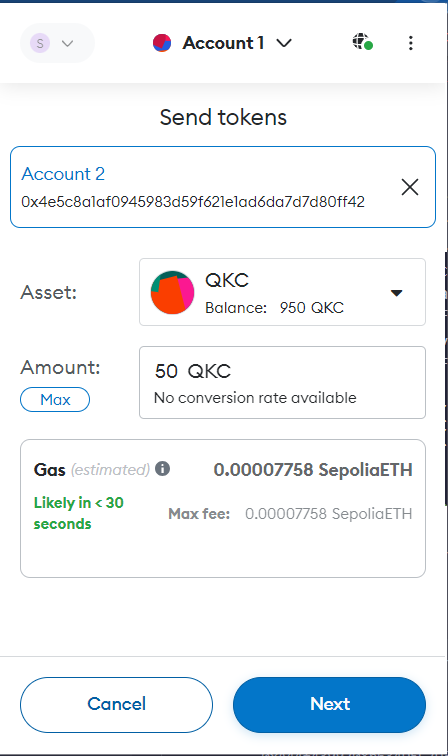
**approve**: This method verifies whether a smart contract is allowed to allocate a certain amount of tokens to a user, considering the total supply.

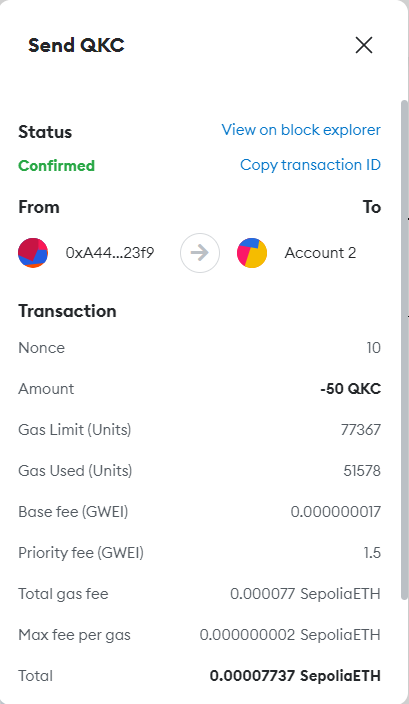
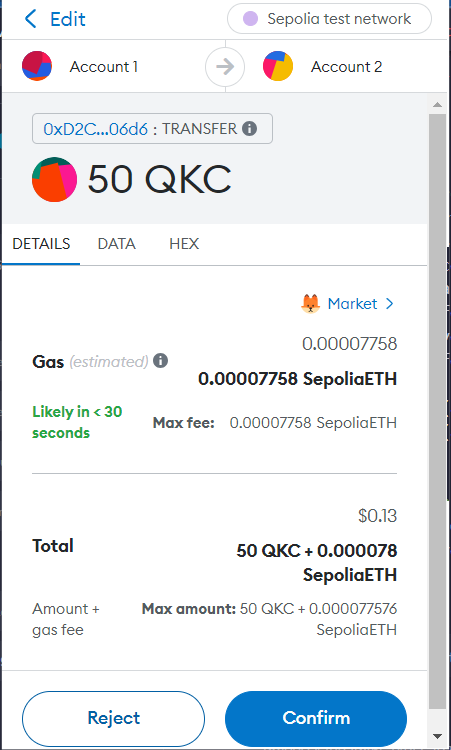


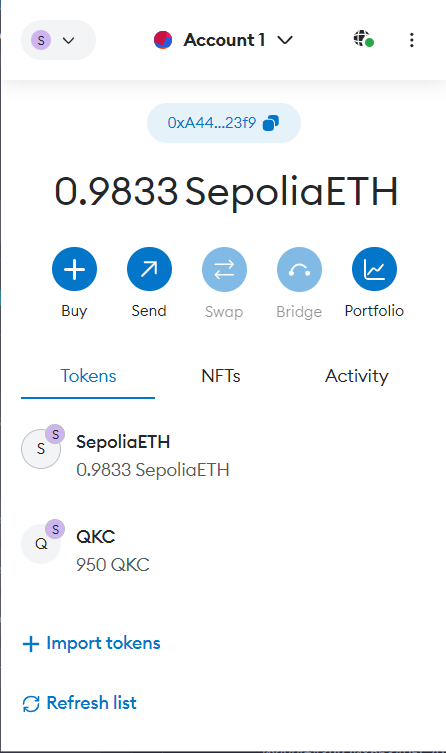
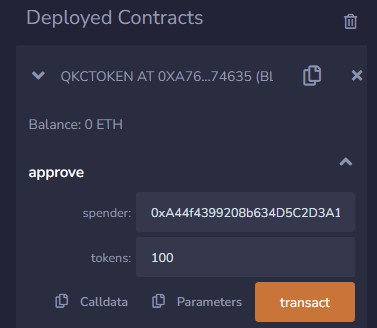
**allowance**: This method is exactly the same as the approved method except that it checks if one user has enough balance to send a certain amount of tokens to another.

# OUTPUT :





Transferred 50 QKC from account 1 to 2

Note : Only 100 QKC can be approved at a time

**Conclusion:** We have succesfully created smart contract and embebeded wallet to perform the transactions.