# Activity 4: ERC-20 token.

## EIP- Ethereum Improvement Proposals and Ethereum Request for Comments

Ethereum Improvement Proposals (EIPs) describe standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards. It includes ERC (Ethereum Request for Comments) for Application-level standards and conventions, including contract standards such as token standards.

## ERC-20 token.

ERC-20 is a standard for digital asset (currency, bonus points).

Tokens can be exchanged through smart contracts.

Simple to deploy.

Accepted by many cryptocurrency wallets, most Ethereum contracts are ERC-20 compliant.

## ERC-20 Token specification.

Token creator must define **fields**:

Token name,

Token symbol,

Number of Tokens created,

Subdivisions

ERC – 20 standard defines **6 functions** which developers must implement:

TotalSupply, BalanceOf, transfer, transferFrom, approve, allowance.

These functions allow wallet app to interrogate user’s balance or transfer tokens to another user.

**function totalSupply()**

**public view returns (uint256);**

**function balanceOf(address tokenOwner)**

**public view returns (uint);**

**function allowance(address tokenOwner, address spender) public view returns (uint);**

**function transfer(address to, uint tokens)**

**public returns (bool);**

**function approve(address spender, uint tokens)**

**public returns (bool);**

**function transferFrom(address from, address to, uint tokens)**

**public returns (bool);**

The **events** defined by ERC-20 are:

**event Approval(address indexed tokenOwner, address indexed spender,**

**uint tokens);**

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

**Step 1:** Define fields:

uint256 nbTokens;

mapping(address => uint256) balances;

mapping(address => mapping (address => uint256)) spendlimit;

string public name = 'Token optional BC';

uint8 public decimals = 0;

string public symbol = 'TOP';

**Step 2:** Define events and modifiers:

event Approval(address indexed tokenOwner, address indexed spender,

uint tokens);

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

modifier checkBalance (address owner, uint tokens) {

        require(tokens <= balances[owner], 'Insufficient funds!');

        \_;

}

modifier checkApproval (address owner, address delegate, uint tokens) {

    require(tokens <= spendlimit[owner][delegate], 'Insufficient allowance!');

        \_;

}

**Step 3**: Set the total number of tokens and set the balance of the owner to the total number of tokens created:

constructor(uint256 tokens) {

   nbTokens = tokens;

   balances[msg.sender] = tokens;

}

**Step 4:** Get total supply:

function totalSupply() public view returns (uint256) {

return nbTokens;

}

**Step 5:** Gat balance for an account:

function balanceOf(address tokenOwner) public view returns (uint) {

    return balances[tokenOwner];

}

**Step 6**: Implement transfer function:

function transfer(address receiver, uint tokens) public checkBalance (msg.sender ,tokens)

returns (bool) {

  balances[msg.sender] = balances[msg.sender] - tokens;

    balances[receiver] = balances[receiver] + tokens;

    emit Transfer(msg.sender, receiver, tokens);

    return true;

}

**Step 7**: Set the number of tokens allowed to be transferred by a delegate.

function approve(address spender, uint tokens)  public returns (bool) {

    spendlimit[msg.sender][spender] = tokens;

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

    return true;

}

**Step 8**: Implement the method that returns the number of tokens allowed to be transferred by a delegate:

function allowance(address tokenOwner, address spender) public view

returns(uint) {

        return spendlimit[tokenOwner][spender];

}

**Step 9**: Implement the functions that transfers from another account, based on the maximum number of tokens allowed for transfer:

function transferFrom(address from, address to, uint tokens)

            public  checkBalance (from, tokens)

                    checkApproval(from, msg.sender, tokens) returns (bool) {

        balances[from] = balances[from] - tokens;

        spendlimit[from][msg.sender] = spendlimit[from][msg.sender]- tokens;

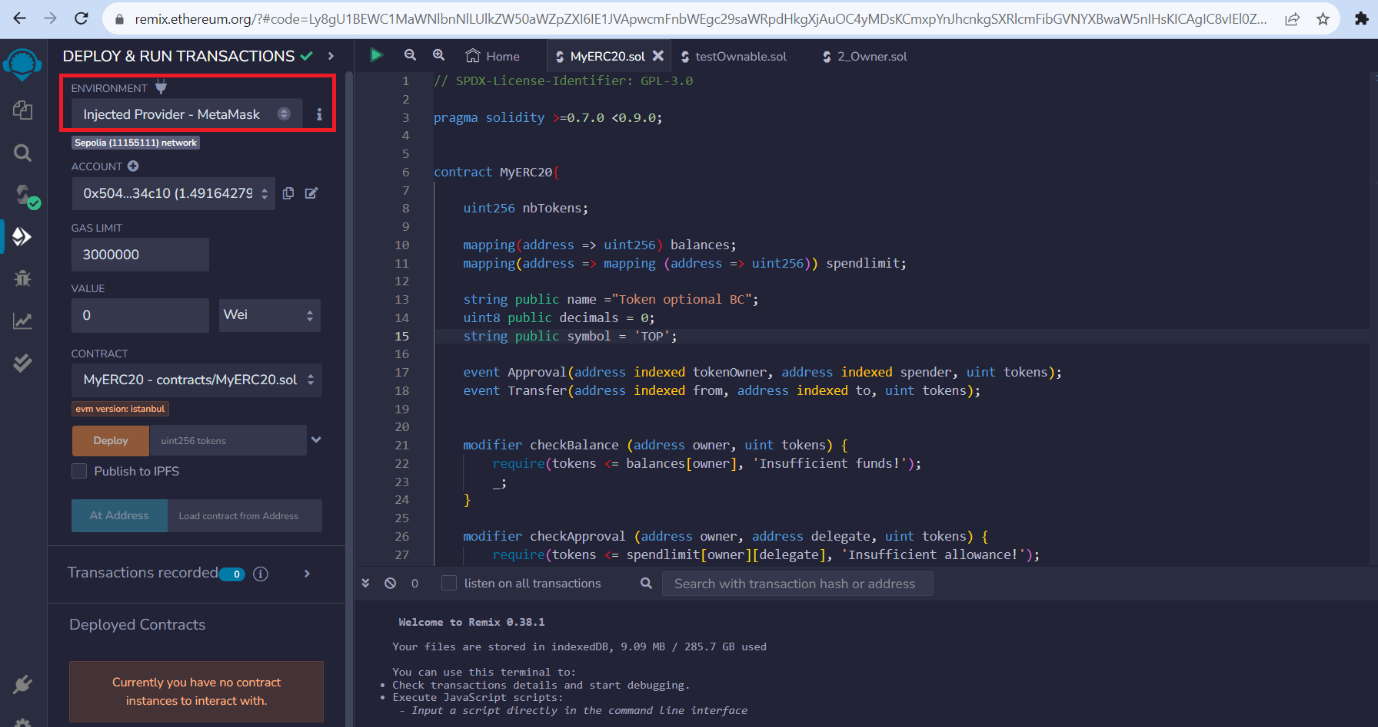
        balances[to] = balances[to] + tokens;

        emit Transfer(from, to, tokens);

        return true;

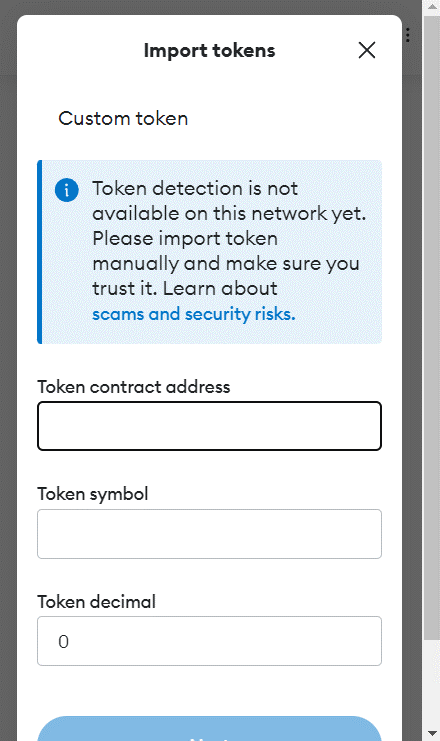
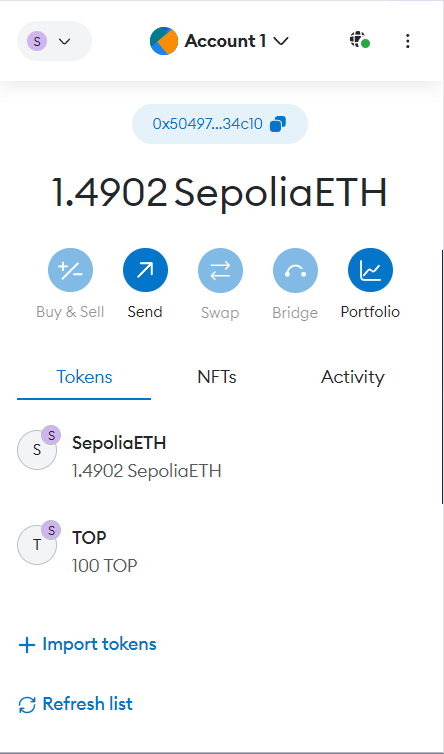
}

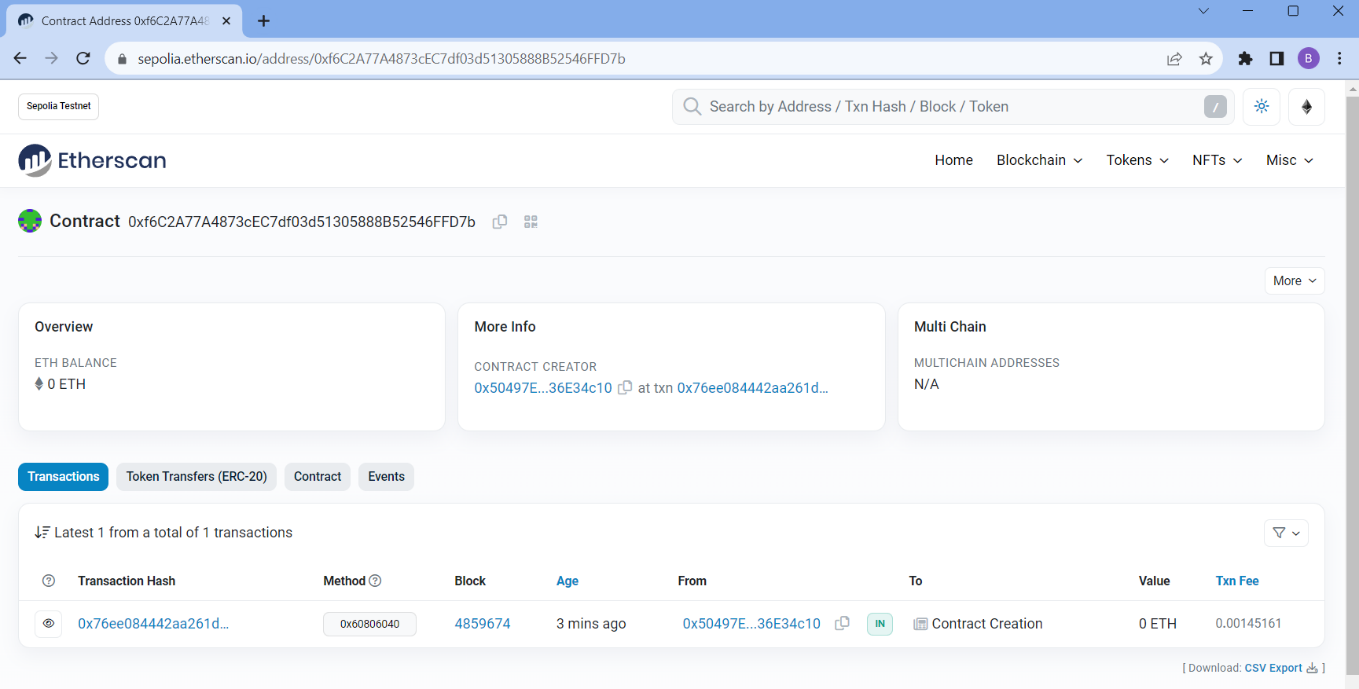
**Step 10**: Deploy on Sepolia network, with truffle or Remix IDE (Injected Web3) and Metamask. Find contract using contract address on EtherScan. Add tokens to Metamask.



Deplyed contract: 0xf6C2A77A4873cEC7df03d51305888B52546FFD7b

0x821496b3B9880b9eA6e7C6aafb0268638501EE5A



**Step 11**: Transfer tokens to another Metamask account.

## Truffle

### npm modules

Check that Truffle is installed:

**>> truffle –version**

Create a new Truffle project in an empty folder:

**>> truffle init**

Install truffle-hdwallet-provider

**>> npm install truffle-hdwallet-provider**

Install dotenv

**>> npm install dotenv**

### File structure

Add in /contract the file MyERC20.sol

Add file migrations/1\_initial\_migration.js

### Infura config

Create an endpoint for Sepolia network with INFURA:

O imagine care conține text, captură de ecran, software, Pagină web

Descriere generată automat

Add file .env and fill it with INFURA MNEMONIC and PROJECT\_ID

MNEMONIC = "…"

PROJECT\_ID = "…"

### Deploy

Modify truffle-config.js:

require('dotenv').config();

const { MNEMONIC, PROJECT\_ID } = process.env;

const HDWalletProvider = require('truffle-hdwallet-provider');

networks: {

sepolia: {

provider: () => new HDWalletProvider(MNEMONIC, `https://sepolia.infura.io/v3/${PROJECT\_ID}`),

network\_id: 11155111, // Sepolia's id

confirmations: 2, // # of confirmations to wait between deployments. (default: 0)

timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)

skipDryRun: true // Skip dry run before migrations? (default: false for public nets )

}

}

Deploy the contract:

>> truffle deploy --network sepolia

**Openzeppelin** framework to write secure smart contracts.

Check IERC20, ERC20

**>> npm install openzeppelin-solidity**

**// SPDX-License-Identifier: GPL-3.0**

**pragma solidity >=0.7.0 <0.9.0;**

**import {ERC20} from "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";**

**contract MyOZERC20 is ERC20 {**

**constructor(string memory name\_, string memory symbol\_, uint256 initialSupply) ERC20(name\_, symbol\_) {**

**\_mint(msg.sender, initialSupply);**

**}**

**}**

truffle deploy --f 2 --network sepolia

Bibliography:

[1] <https://eips.ethereum.org/>

[2] <https://github.com/ethereum/eips/issues/20>