# Lecture 4

# Solidity tutorial,

# ERC–20 token, ERC-721 token

ERC – **Ethereum Requests for Comments** process in Ethereum to define different standards.

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

**ERC-20 token**

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**:

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(unt) {

        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;

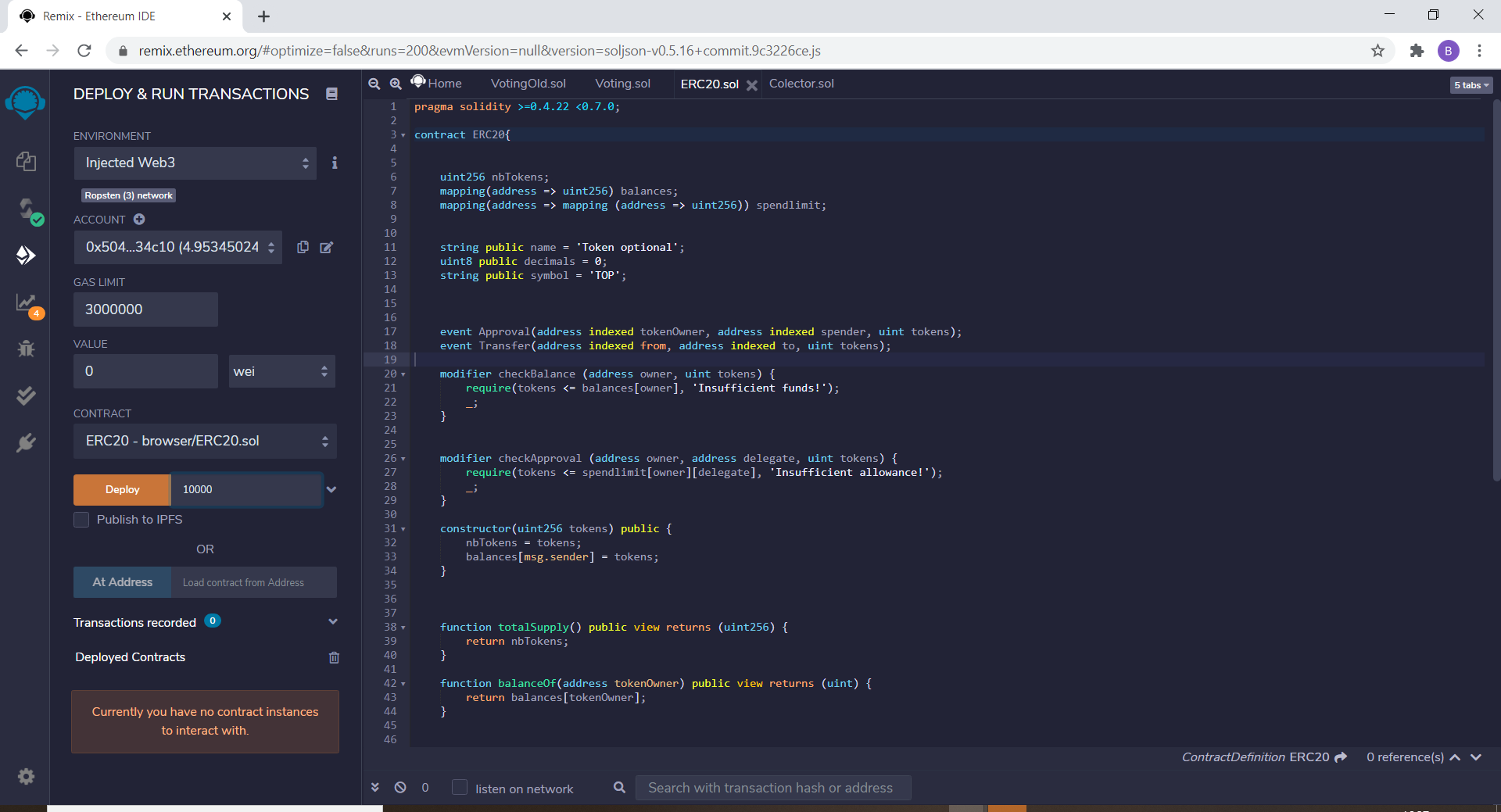
        balances[to] = balances[to] + tokens;

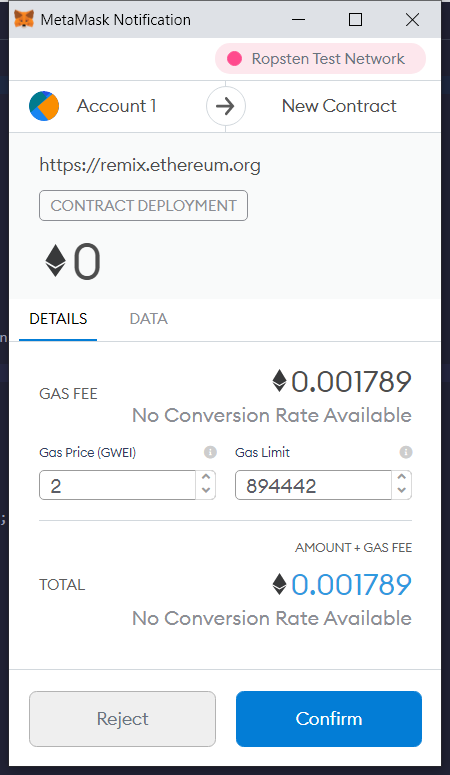
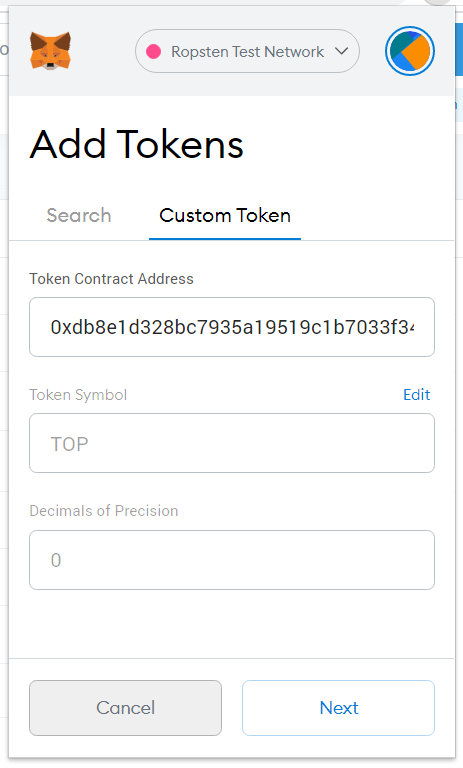
        emit Transfer(from, to, tokens);

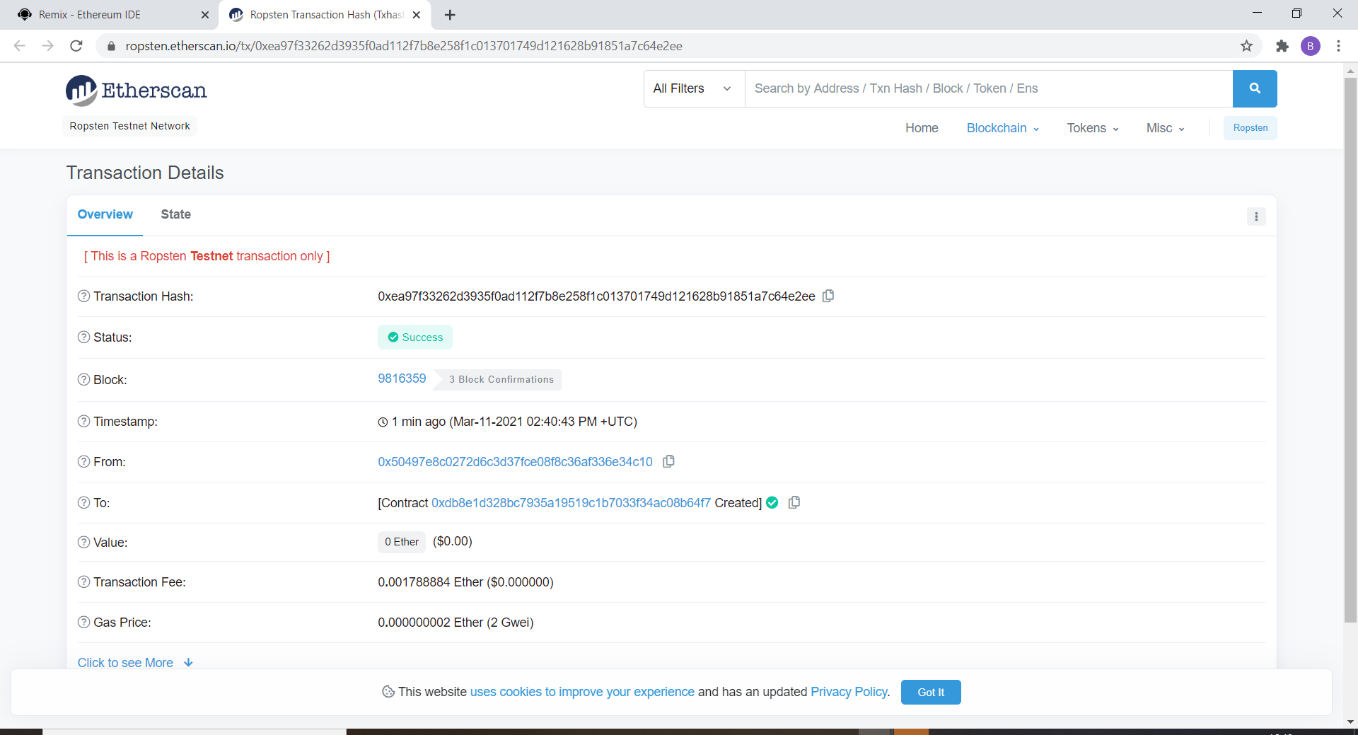
        return true;

}

**Step 10**: Deploy on Ropsten, with truffle or Remix IDE (Injected Web3) and Metamask. Check contract address on Ether Scan. Add tokens to Metamask.





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

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

Check IERC20, ERC20

**>> truffle init**

**>> npm init**

**>> npm install openzeppelin-solidity --save-dev**

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

**contract MyOZERC20 is IERC20, ERC20{**

**constructor (string memory name\_, string memory symbol\_, uint8 decimals\_) ERC20(name\_, symbol\_) public {**

**\_decimals = decimals\_;**

**}**

**}**

## **ERC-721 Token:**

<https://eips.ethereum.org/EIPS/eip-721>

<https://github.com/ethereum/eips/issues/721>

**ERC-721 token**

*Unique digital asset*, assets with unique properties, not interchangeable.

Examples: digital art, collectibles, real estate, items in games, tickets, files, domain names.

ERC-20 tokens are defined by their value. ERC-721 tokens are defined by their properties.

NFT-tokens represent ownership of unique items. Tokens can have only one owner at a time.

NFT-tokens:

* Prevent duplicating items (files), creators can easily claim rights (copy/paste problem).
* Ownership is public and easy to verify. Ownership not controlled by an institution. Creators collect royalties.
* Global marketplace.

**ERC-721 Token**:

* Each token has a unique identifier.
* Each token has a unique owner.
* Each token has a creator. The creator may collect royalties any time the token is sold. Also, the creator can decide how many replicas exist (examples tickets).
* Tokens are not interchangeable.
* Tokens can be bought and sold on NFT – market.

Creating of NFTs (“minting”) and destruction NFTs (“burning”) is not included in the ERC-721 specification.

**Popular ERC-721 tokens** **and applications**:

* **Ethereum Name Service** uses NFT to provide names for Ethereum addresses. (for example, mywallet.eth)

<https://ens.domains/>

<https://unstoppabledomains.com/>

* **Decentralized loans:** borrow money for physical items. If borrower doesn’t pay back, the collateral is sent to the lender.
* **Games**

<https://www.cryptokitties.co/>

<https://sorare.com/>

Token creator defines **fields**:

Token name,

Token symbol,

Token URI.

## Optional interfaces

interface ERC721Metadata {

function name() external view returns (string name);

function symbol() external view returns (string symbol);

function tokenURI(uint256 tokenId) external view returns (string);

}

interface ERC721Enumerable {

function totalSupply() external view returns (uint256);

function tokenByIndex(uint256 \_index) external view returns (uint256);

function tokenOfOwnerByIndex(address \_owner, uint256 \_index) external view returns (uint256);

}

## Functions

ERC – 721 standard defines **9 functions** which developers must implement:

balanceOf, ownerOf, safeTransferFrom, transferFrom, approve, setApprovalForAll, getApproved, isApprovedForAll.

These functions allow contract to keep track of the created tokens.

### **Getters:**

function balanceOf(address \_owner) external view returns (uint256);

returns the number of tokens owned by *\_owner*.

function ownerOf(uint256 \_tokenId) external view returns (address);

returns the address of the owner of token with id *\_tokenId*.

### **Transfer functions:**

function safeTransferFrom(address \_from, address \_to, uint256 \_tokenId)

external payable;

transfers token with *\_tokenId* to the new owner *\_to*. Transfer succeeds if \_from is the owner or if \_from is approved for *\_tokenId*.

The receiver \_to is not a smart contract or it is smart contract implementing ERC721TokenReceiver. If \_to is a smart contract *safeTransferFrom* calls *\_to. onERC721Received*.

function safeTransferFrom(address \_from, address \_to, uint256 \_tokenId, bytes

data) external payable;

extra data *bytes data*.

function transferFrom(address \_from, address \_to, uint256 \_tokenId) external

payable;

doesn’t invoke receiver *\_to.onERC721Received*.

### **Approvals:**

function approve(address \_approved, uint256 \_tokenId) external payable;

approve *\_approved* to transfer *\_tokenId* owned by sender.

function setApprovalForAll(address \_operator, bool \_approved) external;

enable or disable approval for *\_operator* to transfer all token owned by sender.

function getApproved(uint256 \_tokenId) external view returns (address);

get the approved addresses for *\_tokenId*.

function isApprovedForAll(address \_owner, address \_operator) external view

returns (bool);

returns true if *\_operator* is approved for all tokens owned by *\_owner*, false otherwise.

The **events** defined by ERC-721 are:

event Transfer(address indexed \_from, address indexed \_to, uint256 indexed

\_tokenId);

event Approval(address indexed \_owner, address indexed \_approved, uint256 indexed \_tokenId);

event ApprovalForAll(address indexed \_owner, address indexed \_operator, bool \_approved);

**Step 1:** Define fields and getters for *\_name* and *\_symbol*:

string private \_name;

string private \_symbol;

mapping (uint256 => address) private \_owners;

mapping (address => uint256) private \_balances;

mapping (uint256 => address) private \_tokenApprovals;

mapping (address => mapping (address => bool)) private \_operatorApprovals;

function name() external view returns (string memory){

      return \_name;

}

function symbol() external view returns (string memory){

      return \_symbol;

}

**Step 2:** Define events and constructor:

event Transfer(address indexed from, address indexed to,

uint256 indexed tokenId);

event Approval(address indexed owner, address indexed approved,

uint256 indexed tokenId);

event ApprovalForAll(address indexed owner, address indexed operator,

bool approved);

constructor (string memory name\_, string memory symbol\_){

        \_name = name\_;

        \_symbol = symbol\_;

}

**Step 3:** Implement balanceOf and ownerOf methods:

function ownerOf(uint256 tokenId) public view  returns (address) {

        address owner = \_owners[tokenId];

        require(owner != address(0), "owner query for nonexistent token");

        return owner;

    }

function balanceOf(address owner) public view returns (uint256) {

        require(owner != address(0), "balance query for the zero address");

        return \_balances[owner];

    }

**Step 4**: Add a function to verify to existence of token with *tokenId*:

function \_exists(uint256 tokenId) internal view returns (bool) {

        return \_owners[tokenId] != address(0);

}

**Step 5**: Define approval getters and setters:

function isApprovedForAll(address owner, address operator) public view

returns (bool) {

        return \_operatorApprovals[owner][operator];

    }

function getApproved(uint256 tokenId) public view returns (address) {

        require(\_exists(tokenId), "approved query for nonexistent token");

        return \_tokenApprovals[tokenId];

}

function approve(address to, uint256 tokenId) internal {

        \_tokenApprovals[tokenId] = to;

        emit Approval(ownerOf(tokenId), to, tokenId);

}

function setApprovalForAll(address operator, bool approved) public  {

        require(operator != msg.sender, "approve to caller");

        \_operatorApprovals[msg.sender][operator] = approved;

        emit ApprovalForAll(msg.sender, operator, approved);

}

**Step 6:** Declare functions that verify restrictions for senders:

function \_isApprovedOrOwner(address spender, uint256 tokenId) internal view

returns (bool) {

        require(\_exists(tokenId), "operator query for nonexistent token");

        address owner = ownerOf(tokenId);

        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner, spender));

    }

**Step 7:** A receiver contract must implement interface *IERC721Receiver*:

interface IERC721Receiver {

    function onERC721Received(address operator, address from, uint256 tokenId, bytes calldata data) external returns (bytes4);

}

**Step 8:** Declare functions that verify restrictions for receivers:

function \_checkOnERC721Received(address from, address to, uint256 tokenId,

bytes memory \_data)

        private returns (bool)

    {

        if (\_isContract(to)) {

            try IERC721Receiver(to).onERC721Received(msg.sender, from, tokenId, \_data) returns (bytes4 retval) {

                return retval == IERC721Receiver(to).onERC721Received.selector;

            } catch (bytes memory reason) {

               revert("transfer to non ERC721Receiver implementer");

            }

    } else {

        return true;

}

}

function \_isContract(address \_addr) private view returns (bool isContract){

        uint32 size;

        assembly {

            size := extcodesize(\_addr)

        }

        return (size > 0);

}

**Step 9**: Implement transfer function:

function \_transfer(address from, address to, uint256 tokenId) internal {

        require(ownerOf(tokenId) == from,"transfer of token that is not own");

        require(to != address(0), "transfer to the zero address");

        approve(address(0), tokenId);

        \_balances[from] -= 1;

        \_balances[to] += 1;

        \_owners[tokenId] = to;

        emit Transfer(from, to, tokenId);

    }

function \_safeTransfer(address from, address to, uint256 tokenId,

bytes memory \_data) internal {

        \_transfer(from, to, tokenId);

        require(\_checkOnERC721Received(from, to, tokenId, \_data), "transfer to non ERC721Receiver implementer");

}

function transferFrom(address from, address to, uint256 tokenId) public {

        require(\_isApprovedOrOwner(msg.sender, tokenId), "transfer caller is not owner nor approved");

        \_transfer(from, to, tokenId);

    }

function safeTransferFrom(address from, address to, uint256 tokenId) public{

        safeTransferFrom(from, to, tokenId, "");

    }

function safeTransferFrom(address from, address to, uint256 tokenId,

bytes memory \_data) public{

        require(\_isApprovedOrOwner(msg.sender, tokenId), "ERC721: transfer caller is not owner nor approved");

        \_safeTransfer(from, to, tokenId, \_data);

}

**Step 10**: Define mint function:

function \_safeMint(address to, uint256 tokenId, bytes memory \_data) internal{

            \_mint(to, tokenId);

            require(\_checkOnERC721Received(address(0), to, tokenId, \_data), "transfer to non ERC721Receiver implementer");

    }

function \_mint(address to, uint256 tokenId) internal {

        require(to != address(0), "mint to the zero address");

        require(!\_exists(tokenId), "token already minted");

        \_balances[to] += 1;

        \_owners[tokenId] = to;

        emit Transfer(address(0), to, tokenId);

}

**Step 11**: Deploy on Ropsten, with truffle or Remix IDE (Injected Web3) and Metamask. Check contract address on Ether Scan. Add tokens to Metamask.

O imagine care conține text, captură de ecran, monitor, negru

Descriere generată automat

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat

**Step 12**: Transfer tokens to another Metamask account.

## NFT and IPFS

Store large NF-tokens.

IPFS distributed storage network, *content addressability*.

Download IPFS desktop and create account on <https://pinata.cloud/>

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

See ERC721.sol

**>> truffle init**

**>> npm init**

**>> npm install @openzeppelin/contracts**

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

**pragma solidity ^0.8.0;**

**import "@openzeppelin/contracts/token/ERC721/ERC721.sol";**

**import "@openzeppelin/contracts/utils/Counters.sol";**

**contract ERCIPFS is ERC721{**

**constructor() public ERC721("IPFS Asset", "NFIPFS") {}**

**}**

[1] <https://ethereum.org/en/nft/>

[2] <https://etherscan.io/tokens-nft>

[3] <https://ens.domains/>

[4] <https://ethereum.org/en/developers/docs/standards/tokens/erc-721/>

[5] <https://ethereum.org/en/developers/docs/standards/tokens/>

[6] <https://soliditydeveloper.com/erc-721>