KAIST School of Computing

Web3@KAIST

Building Web3 Apps to Solve Real Problems

Building Web3 & Blockchain Applications
(CS492 Special Topics in Computer Science)
Spring 2023

Developing NFT and SBT

Lecture 17 (2023-05-10)

Jason Han, Ph.D

Adjunct Professor of KAIST School of Computing

Founder of Ground X & Klaytn

web3classdao@gmail.com

http://web3classdao.xyz/kaist/

Today’s Lecture 17 Overview

- **Lecture Objective**
  - Understanding non-fungible tokens (NFTs) and ERC721
  - Learning differences between ERC20 and ERC721
  - Learning ERC721 interfaces and how to mint ERC721 NFTs
  - Learning IPFS and Pinata
  - Learning multi-token standard (ERC-1155)
  - Learning Soulbound tokens(SBT) (ERC-5192)

- **Lecture will cover**
  - ERC721 and NFTs
  - ERC1155(Semi-fungible) and ERC5192 (SBT)
  - IPFS and Pinata

References for the lecture

- Ultimate Web3, Full Stack Solidity, and Smart Contract Course by Patrick Collins
  - Lesson 14: HardHat NFTs
- Ethereum EIP-721
- Ethereum ERC-721 Tutorial
- What is ERC-721? by thirdweb
- OpenZeppelin ERC721 docs
- OpenZeppelin ERC721 codes
- NFT MINTER TUTORIAL
- What is ERC-1155? by thirdweb
- Ethereum EIP-1155
- OpenZeppelin ERC1155 codes
- What is SBT? by thirdweb
- IPFS Concepts
- IPFS Simply Explained Youtube

A NFT and SBT contracts

Examples from various sites
with some modification

Clone the code here!

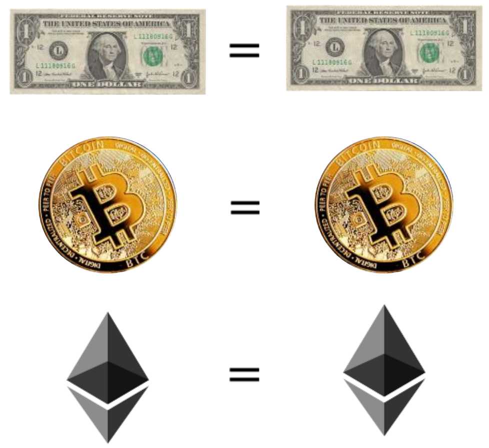
```
git clone https://github.com/web3classdao/nft-sbt.git
```

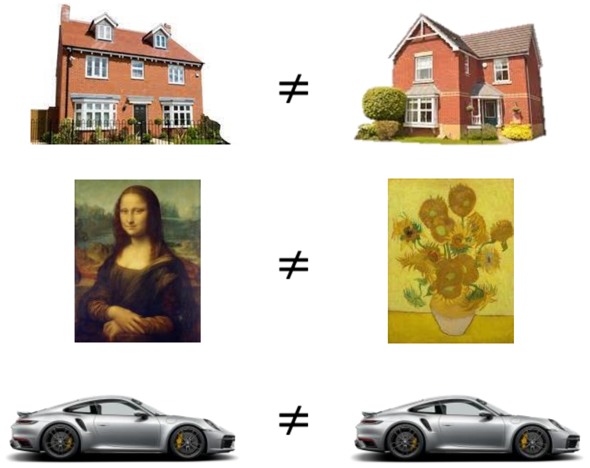
ERC721 NFT Standard

Asset Types

FUNGIBLE

NON-FUNGIBLE





Fungible Token (FT)
ERC-20

Non-Fungible Token (NFT)
ERC-721

ERC721 Token Standard

- **ERC721**: a standard interface(format) for non-fungible assets(NFT) on the Ethereum

  “non-fungible” means they cannot be exchanged on a one-to-one basis due to their unique properties

- **Benefits of ERC721 NFTs**

  - **Standardization**: save time and resources to develop
  - **Interoperability**: easily interact with various wallets, marketplaces, and decentralized applications (dApps) on the Ethereum
  - **Ownership**: allow users to own, transfer, and manage unique digital assets securely
  - **Digital scarcity**: establish digital scarcity with a limited supply and transparent provenance
  - **Programmability**: enable creators to build additional functionalities into their NFTs, such as royalties for artists, in-game utilities, or evolving attributes

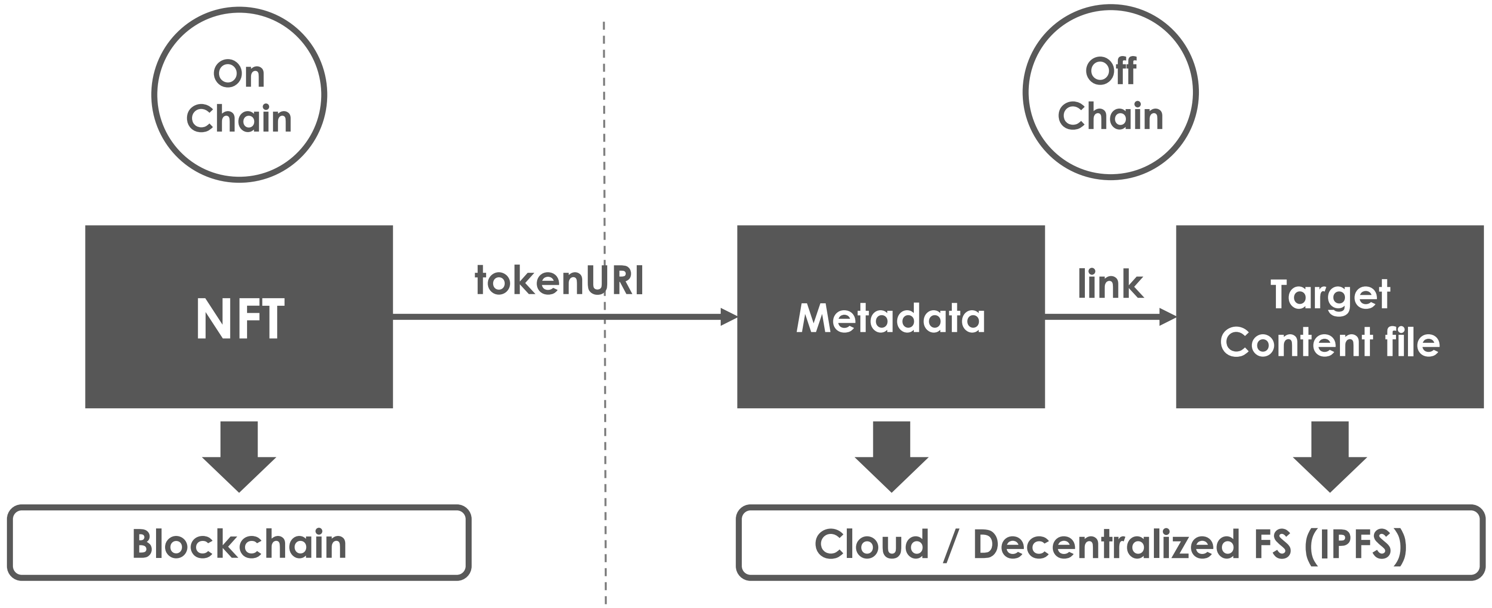
https://blog.thirdweb.com/what-is-erc721-nft/

8 Use Cases for ERC721 NFTs

- **Gaming**: in-game asset such as Axie Infinity, and Gods Unchained
- **Digital Art and Collectibles**: Platforms like OpenSea, Rarible, and Art Blocks allow artists to create and sell their artwork as NFTs
- **Virtual Worlds**: used to represent land parcels, buildings, and other assets in virtual worlds like Decentraland and The Sandbox
- **Domain Names**: Projects like the Ethereum Name Service (ENS) and Unstoppable Domains
- **Event Ticketing**: used to create unique tickets for events
- **Music and Media**: Musicians and other content creators can tokenize their work, allowing fans to purchase and own unique pieces of content
- **Identity and Certification**: used to represent digital identities, educational certificates, or professional licenses
- **DeFi and Financial Instruments**: used to create unique financial instruments, such as tokenized real estate, insurance policies, or bonds in DeFi

https://blog.thirdweb.com/what-is-erc721-nft/

How ERC721 NFTs work

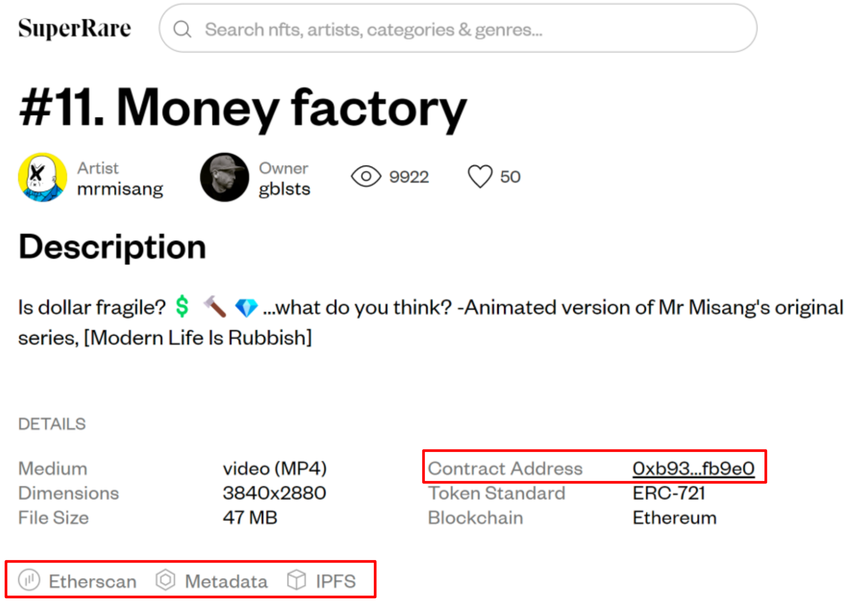


NFT Identifier

The pair *(contract address, uint256 tokenId)*

Note that the content that an NFT points to
is not itself an identifying element of the NFT

ERC721 NFT Example
with metadata & tokenURI, IPFS



SuperRare

Search nfts, artists, categories & genres...

#11. Money factory

Artist mrmisang | Owner gblsts | 9922 | 50

Description

Is dollar fragile? 💲 🔨 💎 ...what do you think? -Animated version of Mr Misang's original series, [Modern Life Is Rubbish]

DETAILS

Medium	video (MP4)	Contract Address	0xb93...fb9e0
Dimensions	3840x2880	Token Standard	ERC-721
File Size	47 MB	Blockchain	Ethereum

Etherscan | Metadata | IPFS

https://superrare.com/artwork-v2/11.-money-factory-23418

ERC721 NFT Example: How it works

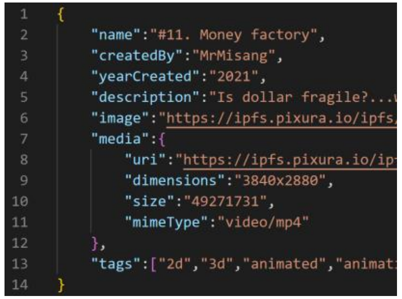
You can check the tx of minting the NFT
https://etherscan.io/tx/0x29360be4221f46799ef10fe1648799c13fa3b99837263805f4d0eb5bed403f91

NFT Identifier

ERC721 contract address
0xb932a70A57673d89f4acfFBE830E8ed7f75Fb9e0

Token ID: *23418*

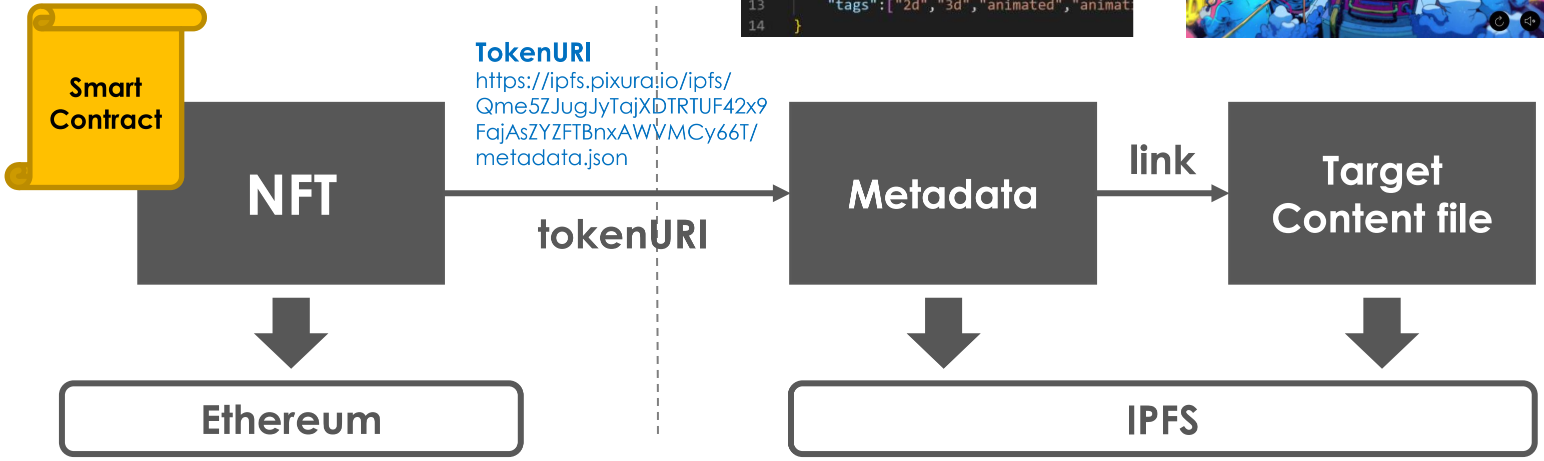
metadata.json



```
{
    "name":"#11. Money factory",
    "createdBy":"MrMisang",
    "yearCreated":"2021",
    "description":"Is dollar fragile?...
    "image":"https://ipfs.pixura.io/ipfs
    "media":{
        "uri":"https://ipfs.pixura.io/ip
        "dimensions":"3840x2880",
        "size":"49271731",
        "mimeType":"video/mp4"
    },
    "tags":["2d","3d","animated","animat
}
```

MoneyFactoryre.mp4

TokenURI
https://ipfs.pixura.io/ipfs/Qme5ZJugJyTajXDTRTUF42x9FajAsZYZFTBnxAWVMCy66T/metadata.json



Data Structure: ERC20 vs. ERC721

ERC20 Token

Manage only which addresses hold how many tokens

balances

address	amount
addr1	100
addr2	50
addr3	200
●	●
addr10	20

ERC721 NFT

Manage who owns each token and how many tokens each address holds

owners

tokenId	address
1	addr1
2	addr7
3	addr4
●	●
100	addr2

balances

address	amount
addr1	1
addr2	2
addr3	1
●	●
addr10	5

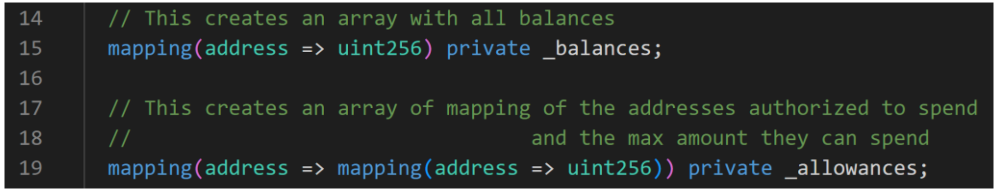
tokenUris

tokenId	tokenUri
1	uri1
2	uri2
3	uri3
●	●
100	uri100

ERC721 NFT is more complicated

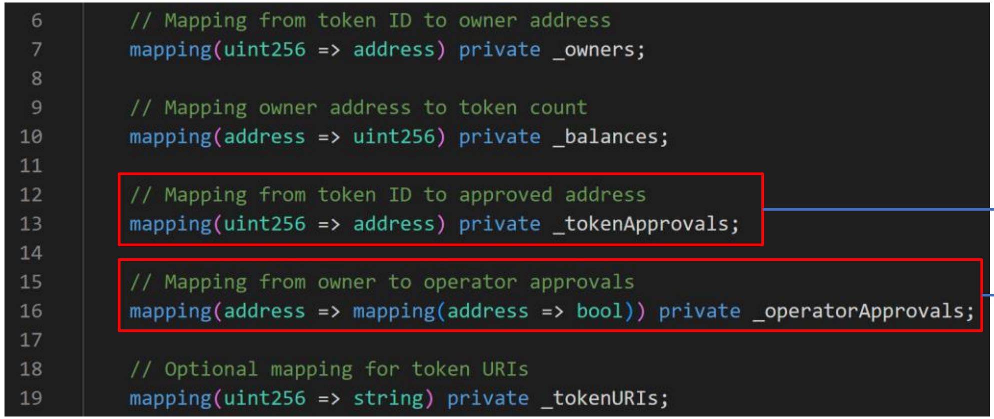
Data Structure: ERC20 vs. ERC721

ERC20 Token



```
// This creates an array with all balances
mapping(address => uint256) private _balances;

// This creates an array of mapping of the addresses authorized to spend
//                                    and the max amount they can spend
mapping(address => mapping(address => uint256)) private _allowances;
```

ERC721 NFT

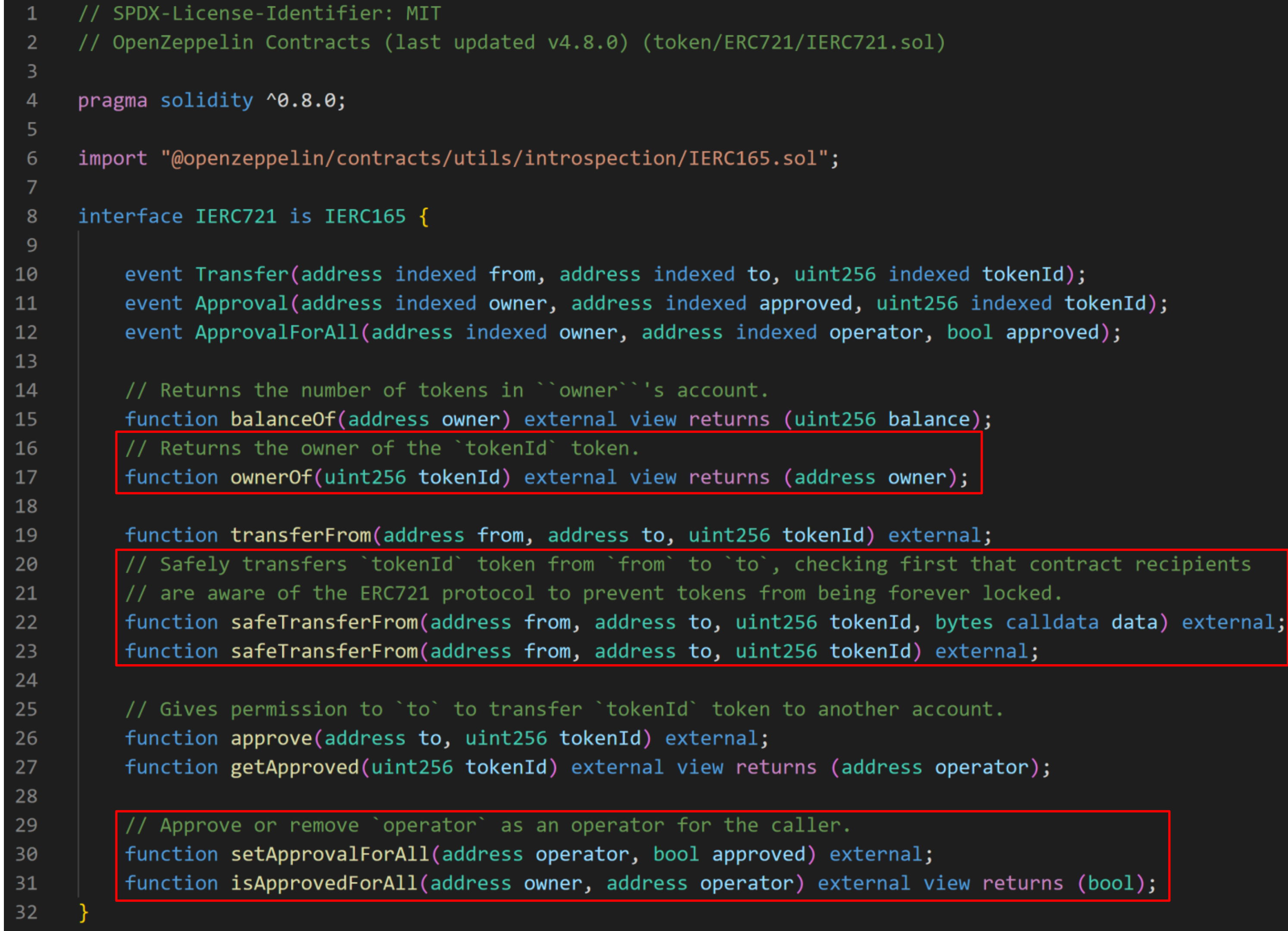


```
// Mapping from token ID to owner address
mapping(uint256 => address) private _owners;

// Mapping owner address to token count
mapping(address => uint256) private _balances;

// Mapping from token ID to approved address
mapping(uint256 => address) private _tokenApprovals;

// Mapping from owner to operator approvals
mapping(address => mapping(address => bool)) private _operatorApprovals;

// Optional mapping for token URIs
mapping(uint256 => string) private _tokenURIs;
```

Approval for each token

Approval for all tokens which an account holds

ERC721 Interface



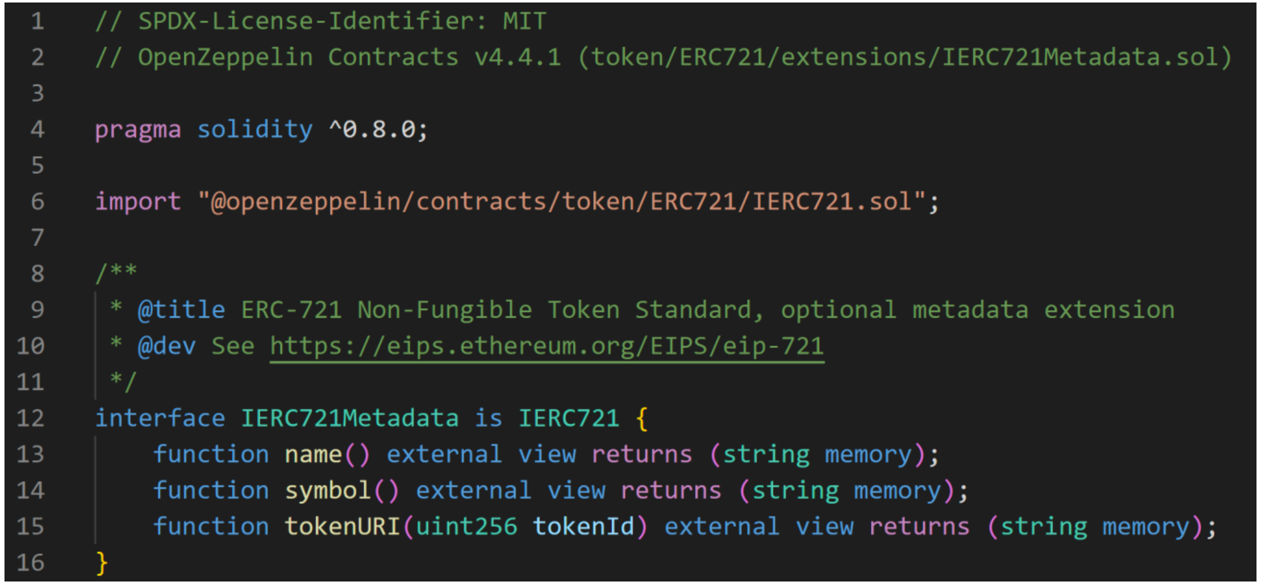
```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (token/ERC721/IERC721.sol)

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/introspection/IERC165.sol";

interface IERC721 is IERC165 {

    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);

    // Returns the number of tokens in ``owner``'s account.
    function balanceOf(address owner) external view returns (uint256 balance);
    // Returns the owner of the `tokenId` token.
    function ownerOf(uint256 tokenId) external view returns (address owner);

    function transferFrom(address from, address to, uint256 tokenId) external;
    // Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
    // are aware of the ERC721 protocol to prevent tokens from being forever locked.
    function safeTransferFrom(address from, address to, uint256 tokenId, bytes calldata data) external;
    function safeTransferFrom(address from, address to, uint256 tokenId) external;

    // Gives permission to `to` to transfer `tokenId` token to another account.
    function approve(address to, uint256 tokenId) external;
    function getApproved(uint256 tokenId) external view returns (address operator);

    // Approve or remove `operator` as an operator for the caller.
    function setApprovalForAll(address operator, bool approved) external;
    function isApprovedForAll(address owner, address operator) external view returns (bool);
}
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/IERC721.sol

Transfers may be initiated by

- *The owner of an NFT*
- *The approved address of an NFT*
  (required to be set in _tokenApprovals)
- *An authorized operator of the current owner of an NFT*
  (required to be set in _operatorApprovals)

IERC721Metadata Interface

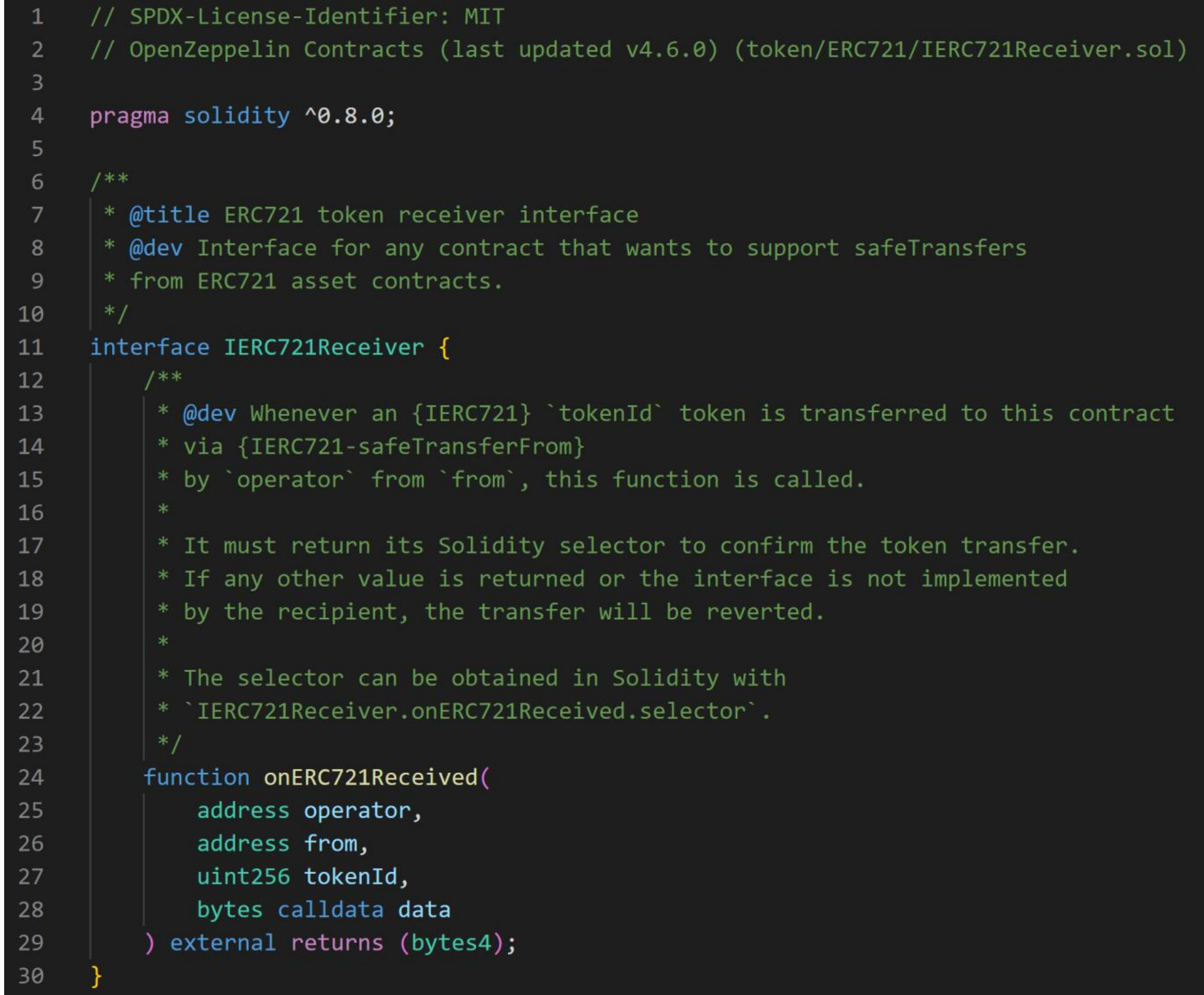


```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC721/extensions/IERC721Metadata.sol)

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";

/**
 * @title ERC-721 Non-Fungible Token Standard, optional metadata extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
interface IERC721Metadata is IERC721 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function tokenURI(uint256 tokenId) external view returns (string memory);
}
```

IERC721Receiver Interface

Any contract that wants to receive ERC721 NFT via safeTransfer() SHOULD implement this interface

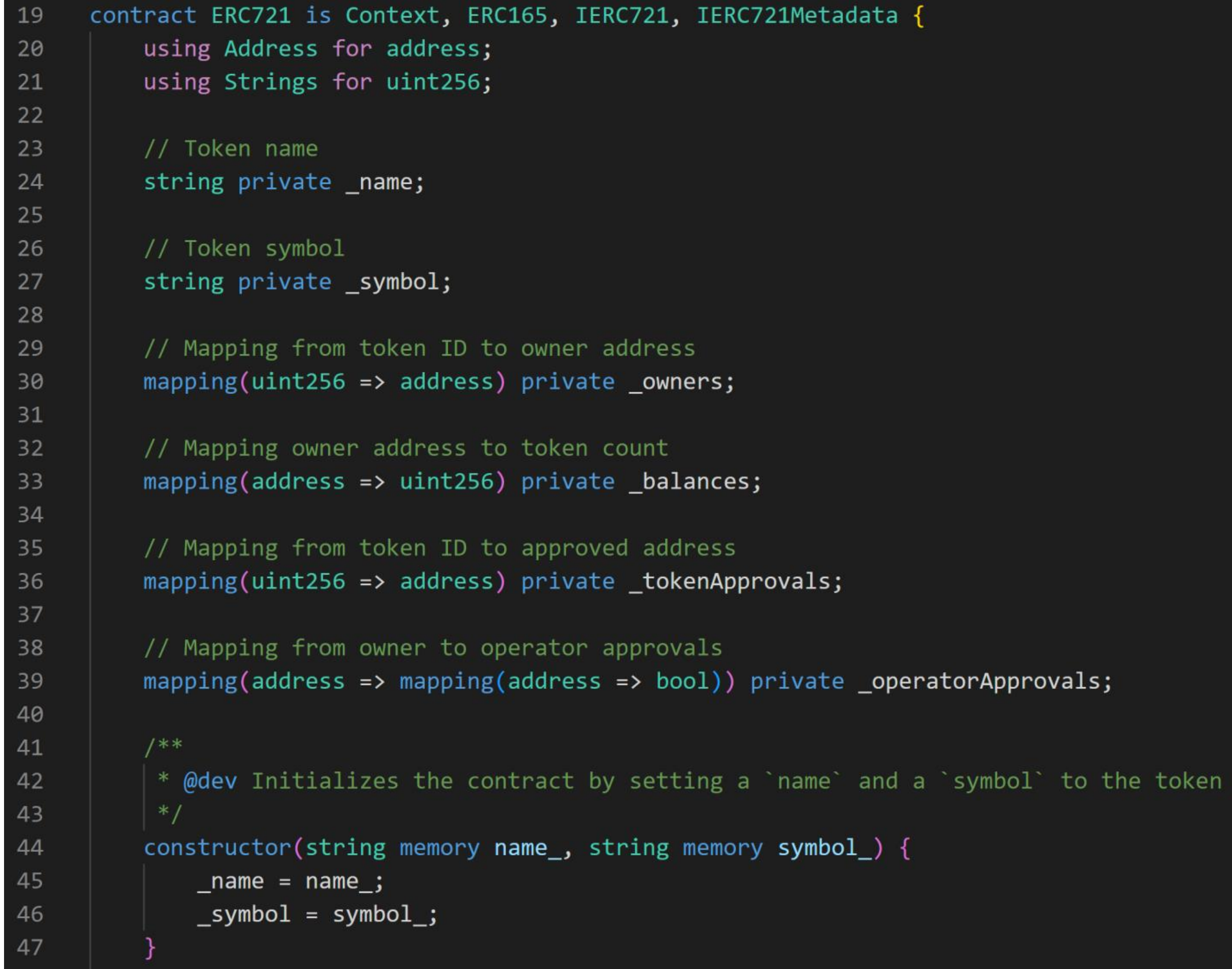
→ *It shows the contract can handle ERC721 NFTs*



```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC721/IERC721Receiver.sol)

pragma solidity ^0.8.0;

/**
 * @title ERC721 token receiver interface
 * @dev Interface for any contract that wants to support safeTransfers
 * from ERC721 asset contracts.
 */
interface IERC721Receiver {
    /**
     * @dev Whenever an {IERC721} `tokenId` token is transferred to this contract
     * via {IERC721-safeTransferFrom}
     * by `operator` from `from`, this function is called.
     *
     * It must return its Solidity selector to confirm the token transfer.
     * If any other value is returned or the interface is not implemented
     * by the recipient, the transfer will be reverted.
     *
     * The selector can be obtained in Solidity with
     * `IERC721Receiver.onERC721Received.selector`.
     */
    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    ) external returns (bytes4);
}
```

OpenZeppelin ERC721 NFT Reference Implementation

ERC721.sol



```
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the token
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }
```

You can easily understand most codes of ERC721.sol since it's similar to ERC20.sol Please look at the source code

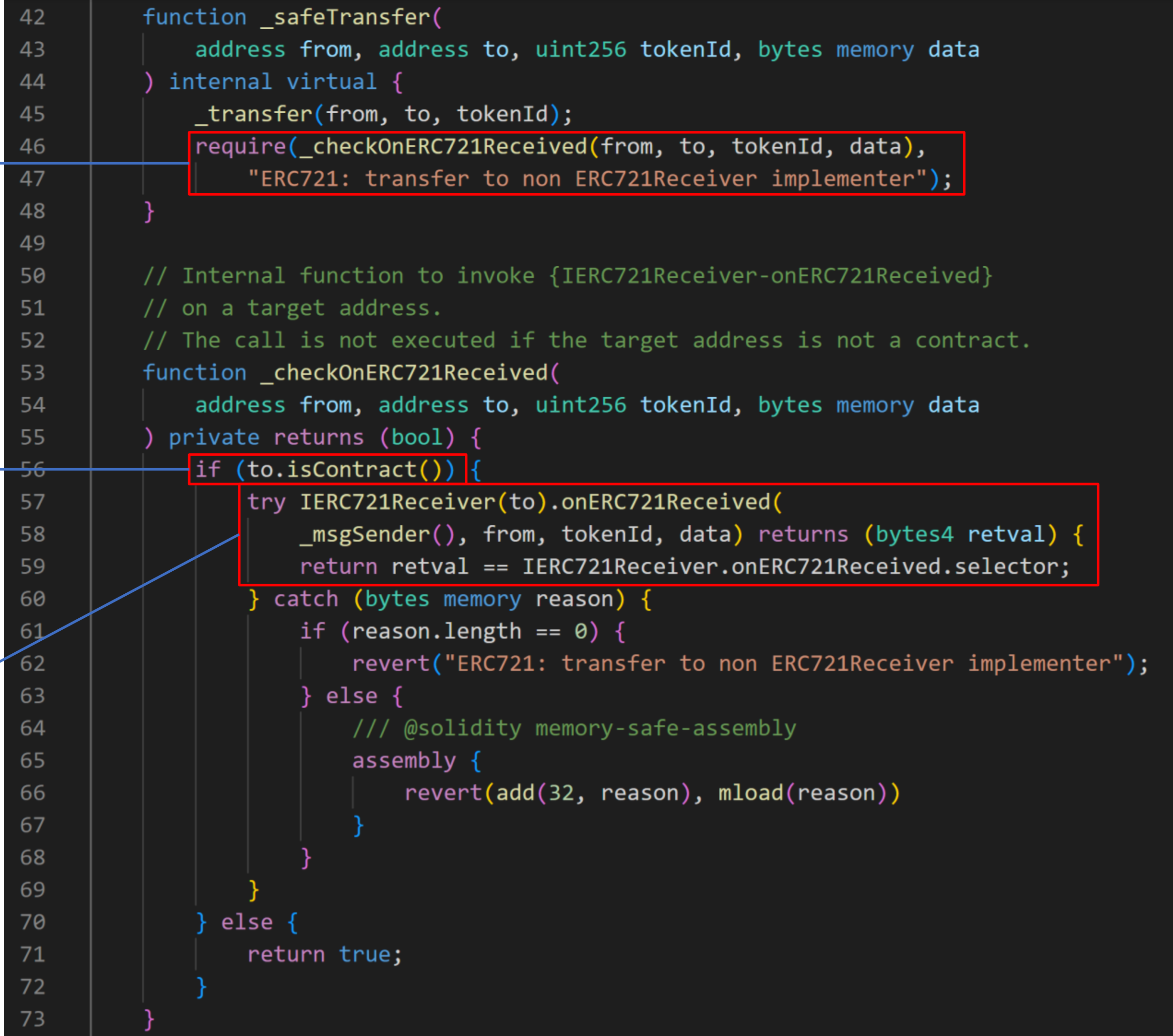
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol *safeTransfer() checks to see if the receiving contract can handle ERC721 NFTs with _checkOnERC721Received()*

If the receiving address is not a contract, the call is not executed

call The receiving contract's onERC721Received()

If the selector of onERC721Received() is returned, the call is successfully passed

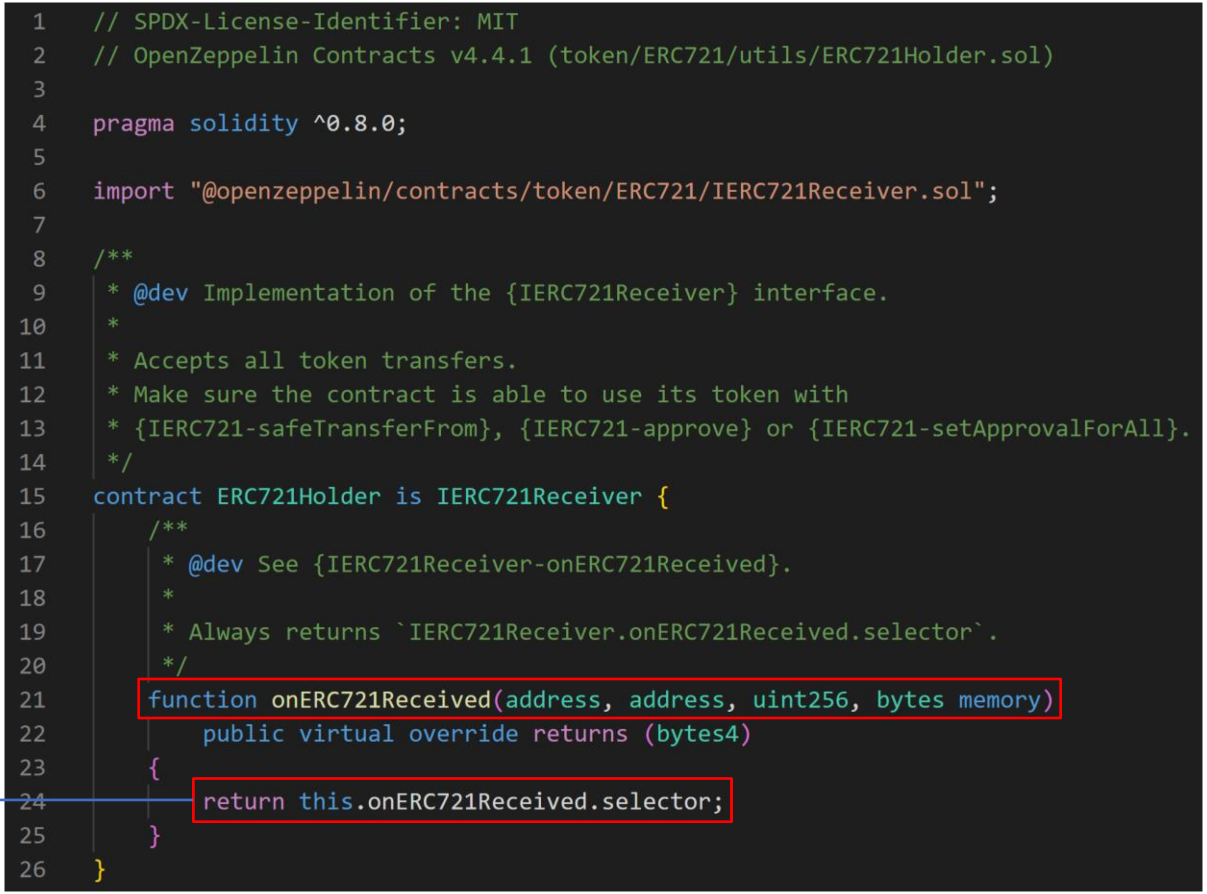
The receiving contract should implement onERC721Received() to return the selector



```
function _safeTransfer(
    address from, address to, uint256 tokenId, bytes memory data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, data),
        "ERC721: transfer to non ERC721Receiver implementer");
}

// Internal function to invoke {IERC721Receiver-onERC721Received}
// on a target address.
// The call is not executed if the target address is not a contract.
function _checkOnERC721Received(
    address from, address to, uint256 tokenId, bytes memory data
) private returns (bool) {
    if (to.isContract()) {
        try IERC721Receiver(to).onERC721Received(
            _msgSender(), from, tokenId, data) returns (bytes4 retval) {
            return retval == IERC721Receiver.onERC721Received.selector;
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                revert("ERC721: transfer to non ERC721Receiver implementer");
            } else {
                /// @solidity memory-safe-assembly
                assembly {
                    revert(add(32, reason), mload(reason))
                }
            }
        }
    } else {
        return true;
    }
}
```

ERC721Holder: Ref. Implementation of IERC721Receiver

You can inherit this contract to allow your contract to receive ERC721 NFTs through safeTransfer()

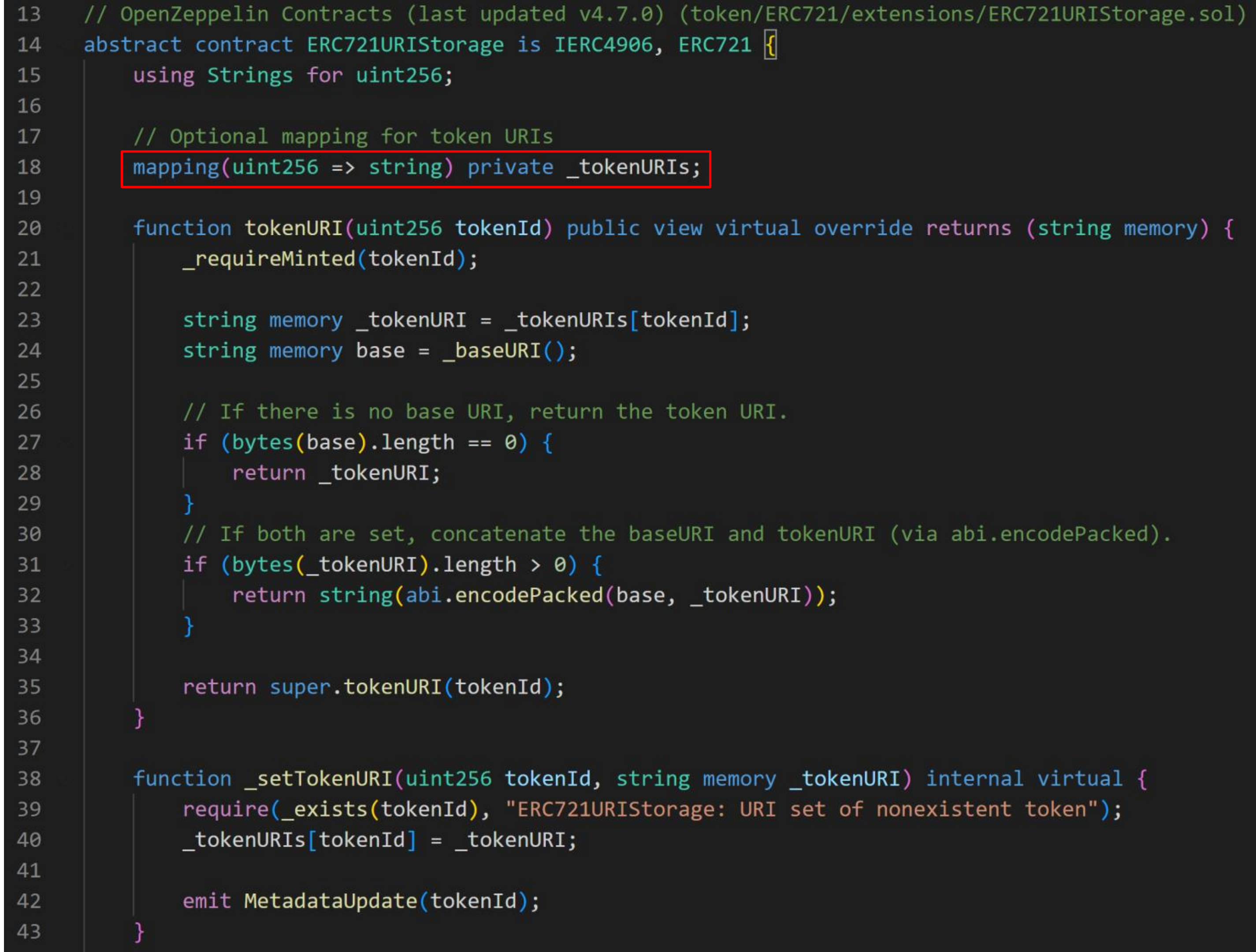


```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC721/utils/ERC721Holder.sol)

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";

/**
 * @dev Implementation of the {IERC721Receiver} interface.
 *
 * Accepts all token transfers.
 * Make sure the contract is able to use its token with
 * {IERC721-safeTransferFrom}, {IERC721-approve} or {IERC721-setApprovalForAll}.
 */
contract ERC721Holder is IERC721Receiver {
    /**
     * @dev See {IERC721Receiver-onERC721Received}.
     *
     * Always returns `IERC721Receiver.onERC721Received.selector`.
     */
    function onERC721Received(address, address, uint256, bytes memory)
        public virtual override returns (bytes4)
    {
        return this.onERC721Received.selector;
    }
}
```

return the selector of onERC721Received

OpenZeppelin ERC721URIStorage Reference Impl.

You can inherit this contract to implement ERC721 NFTs with a tokenURI storage

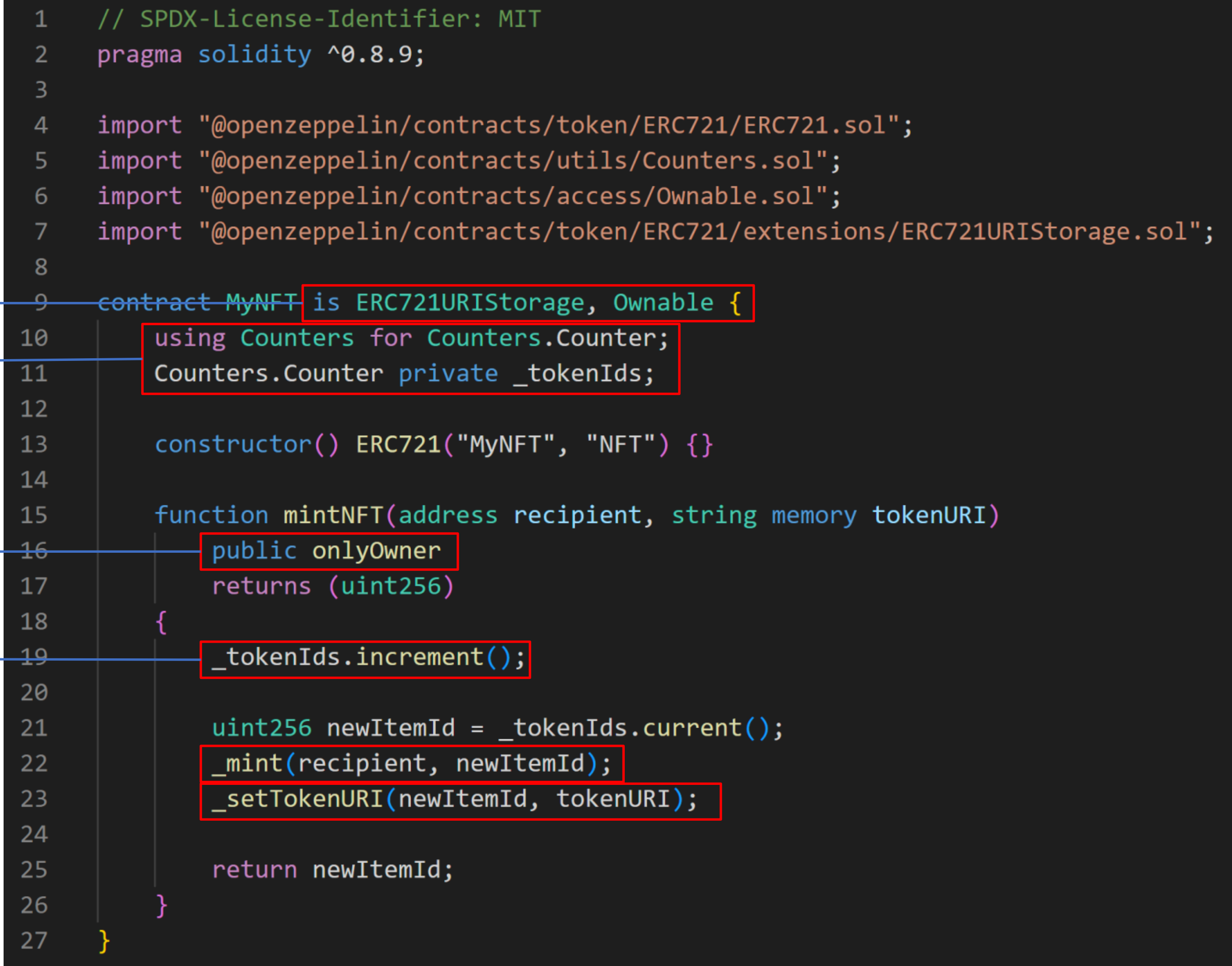


```solidity
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC721/extensions/ERC721URIStorage.sol)
abstract contract ERC721URIStorage is IERC4906, ERC721 {
    using Strings for uint256;

    // Optional mapping for token URIs
    mapping(uint256 => string) private _tokenURIs;

    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        _requireMinted(tokenId);

        string memory _tokenURI = _tokenURIs[tokenId];
        string memory base = _baseURI();

        // If there is no base URI, return the token URI.
        if (bytes(base).length == 0) {
            return _tokenURI;
        }
        // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked).
        if (bytes(_tokenURI).length > 0) {
            return string(abi.encodePacked(base, _tokenURI));
        }

        return super.tokenURI(tokenId);
    }

    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual {
        require(_exists(tokenId), "ERC721URIStorage: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;

        emit MetadataUpdate(tokenId);
    }
```

Minting ERC721 NFT

Minting My ERC721 NFTs

MyNFT.sol



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

contract MyNFT is ERC721URIStorage, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("MyNFT", "NFT") {}

    function mintNFT(address recipient, string memory tokenURI)
        public onlyOwner
        returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```

Inherit ERC721URIStorage

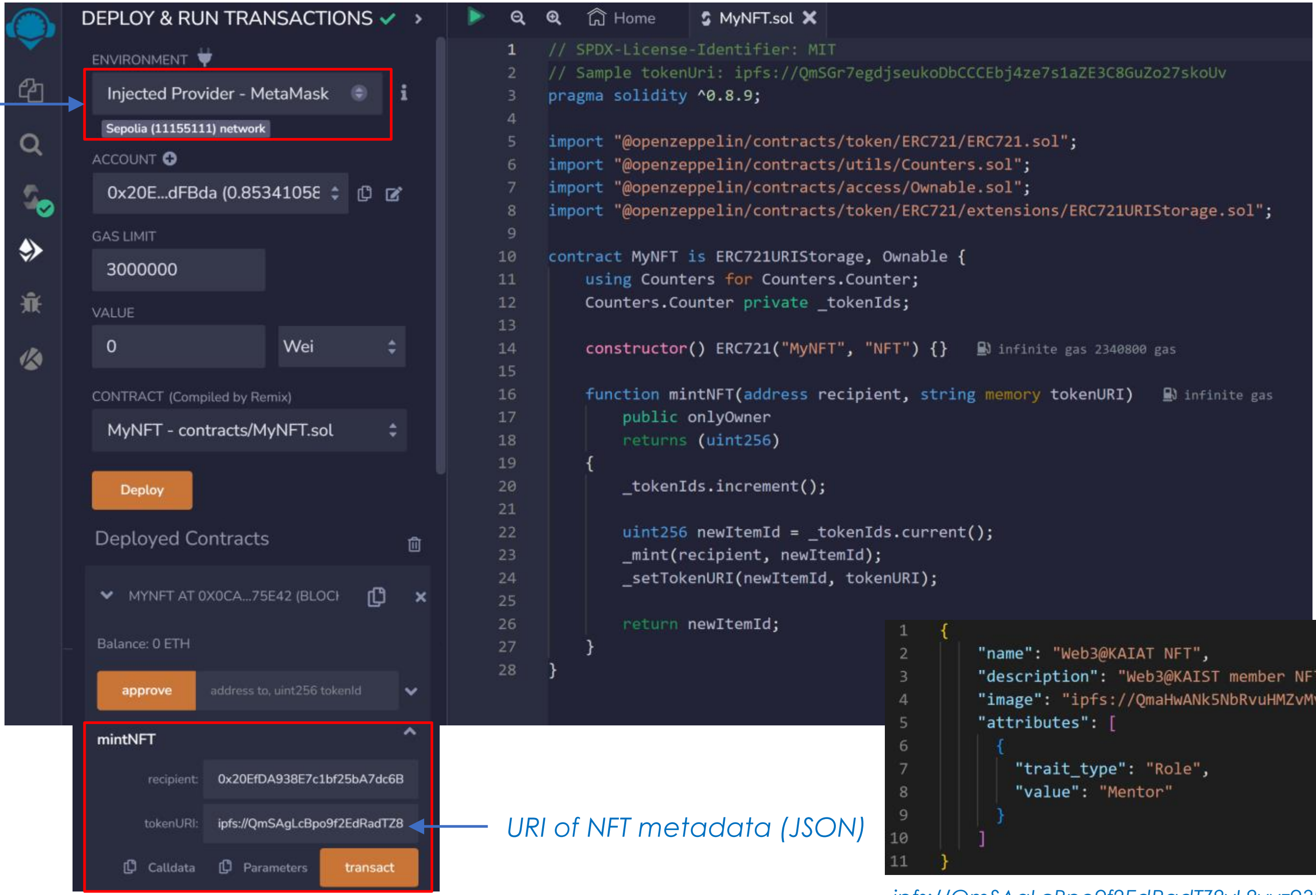
tokenIds
using the Counter library

an owner can mint only
using the Ownable labrary

use Counter function

https://ethereum.org/en/developers/tutorials/how-to-write-and-deploy-an-nft/

Deploy & Run ERC721 Contracts with Remix



using Metamask

ENVIRONMENT: Injected Provider - MetaMask
Sepolia (11155111) network

ACCOUNT: 0x20E...dFBda (0.85341058

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT (Compiled by Remix): MyNFT - contracts/MyNFT.sol

MYNFT AT 0X0CA...75E42 (BLOCI

Balance: 0 ETH

approve: address to, uint256 tokenId

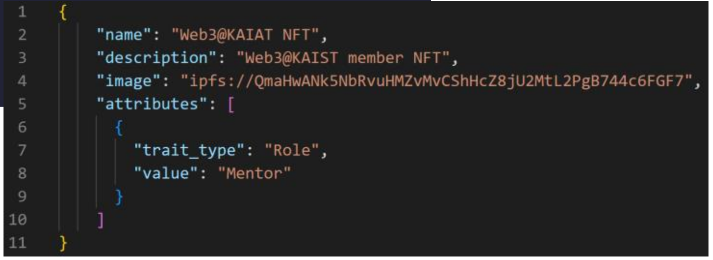
mintNFT

recipient: 0x20EfDA938E7c1bf25bA7dc6B

tokenURI: ipfs://QmSAgLcBpo9f2EdRadTZ8

URI of NFT metadata (JSON)

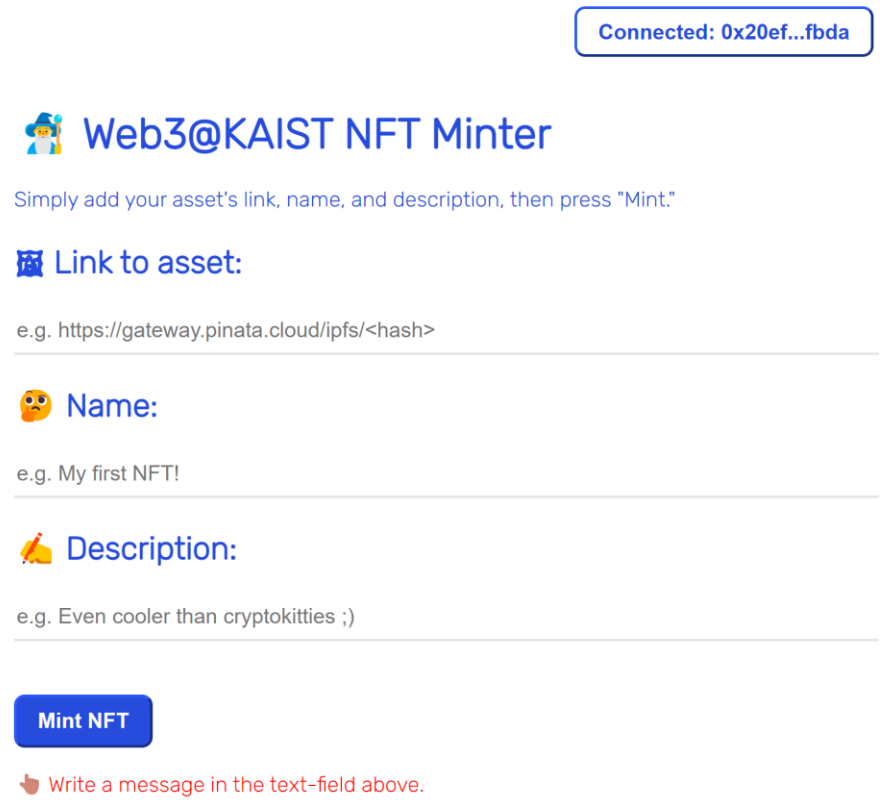
```
// SPDX-License-Identifier: MIT
// Sample tokenUri: ipfs://QmSGr7egdjseukoDbCCCEbj4ze7s1aZE3C8GuZo27skoUv
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

contract MyNFT is ERC721URIStorage, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("MyNFT", "NFT") {}    infinite gas 2340800 gas

    function mintNFT(address recipient, string memory tokenURI)    infinite gas
        public onlyOwner
        returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```



```
{
    "name": "Web3@KAIAT NFT",
    "description": "Web3@KAIST member NFT",
    "image": "ipfs://QmaHwANk5NbRvuHMZvMvCShHcZ8jU2MtL2PgB744c6FGF7",
    "attributes": [
      {
        "trait_type": "Role",
        "value": "Mentor"
      }
    ]
}
```

ipfs://QmSAgLcBpo9f2EdRadTZ8yL8vvz93KGDxkPg3QJUfsqMfY

NFT Minter with React Frontend



Connected: 0x20ef...fbda

🧙 Web3@KAIST NFT Minter

Simply add your asset's link, name, and description, then press "Mint."

🖼 Link to asset:

e.g. https://gateway.pinata.cloud/ipfs/<hash>

🤔 Name:

e.g. My first NFT!

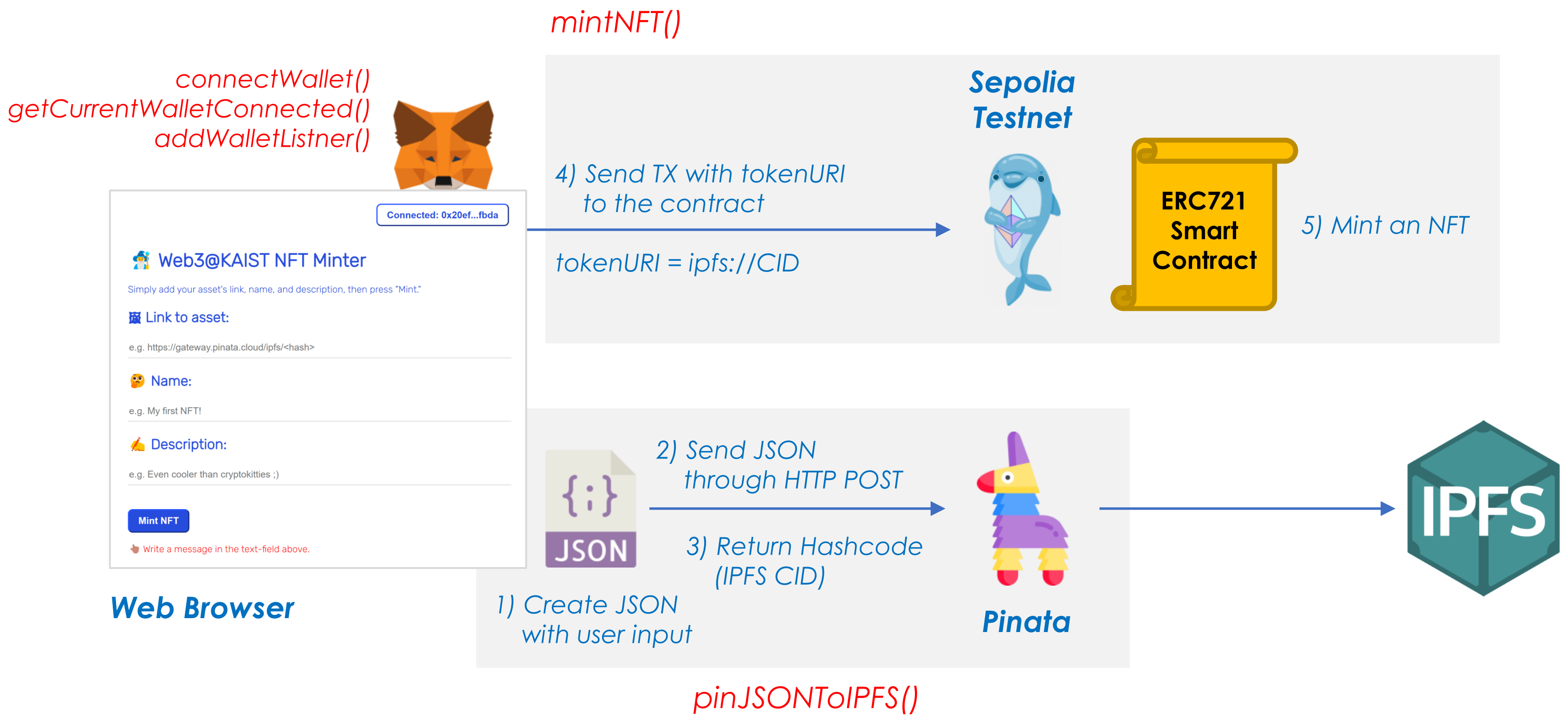
✍️ Description:

e.g. Even cooler than cryptokitties ;)

Mint NFT

👆🏽 Write a message in the text-field above.

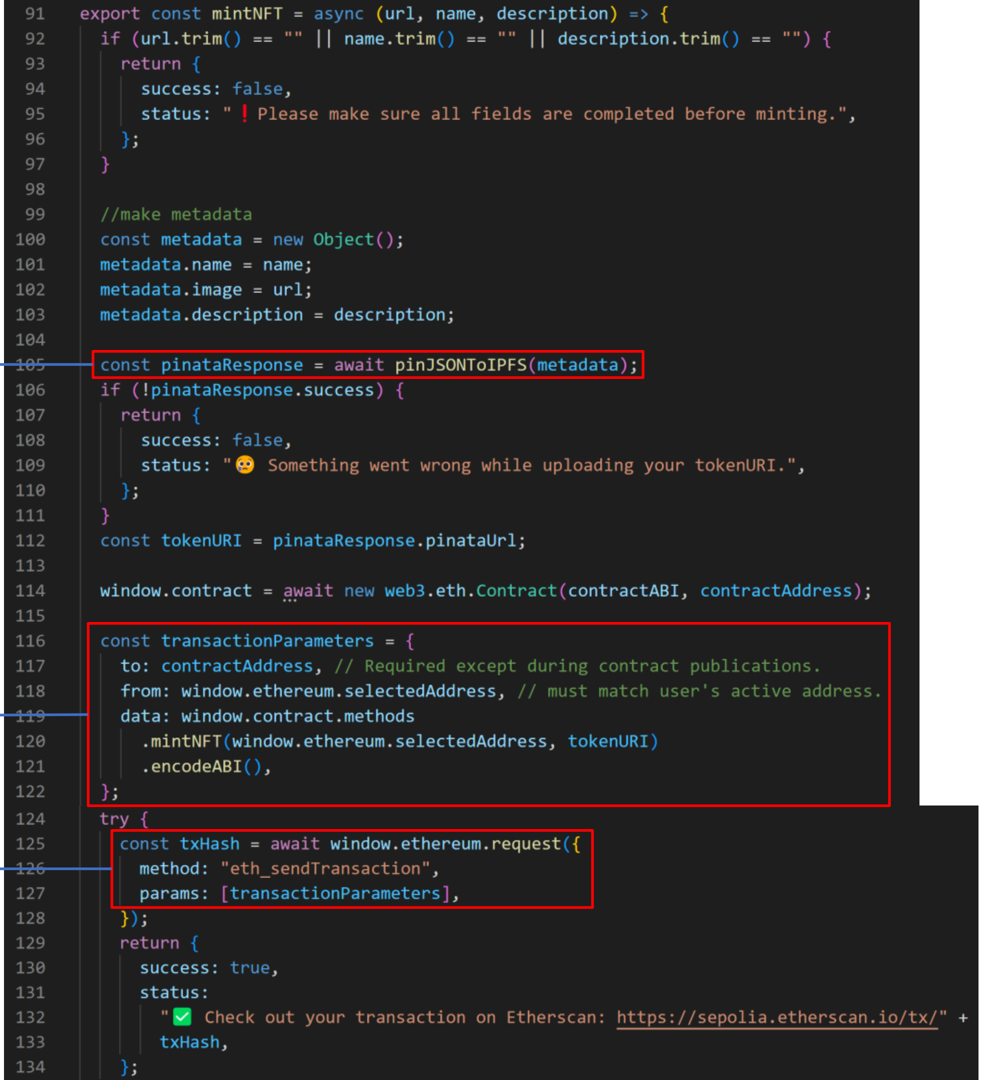
NFT Minter with React Frontend


mintNFT()
connectWallet()
getCurrentWalletConnected()
addWalletListner()
Connected: 0x20ef...fbda
Web3@KAIST NFT Minter
Simply add your asset's link, name, and description, then press "Mint."
Link to asset:
e.g. https://gateway.pinata.cloud/ipfs/<hash>
Name:
e.g. My first NFT!
Description:
e.g. Even cooler than cryptokitties ;)
Mint NFT
Write a message in the text-field above.
Web Browser
Sepolia Testnet
4) Send TX with tokenURI to the contract
tokenURI = ipfs://CID
ERC721 Smart Contract
5) Mint an NFT
JSON
2) Send JSON through HTTP POST
3) Return Hashcode (IPFS CID)
1) Create JSON with user input
Pinata
IPFS
pinJSONToIPFS()

https://ethereum.org/en/developers/tutorials/nft-minter/

React Frontend

interact.js



```
export const mintNFT = async (url, name, description) => {
  if (url.trim() == "" || name.trim() == "" || description.trim() == "") {
    return {
      success: false,
      status: "❗Please make sure all fields are completed before minting.",
    };
  }

  //make metadata
  const metadata = new Object();
  metadata.name = name;
  metadata.image = url;
  metadata.description = description;

  const pinataResponse = await pinJSONToIPFS(metadata);
  if (!pinataResponse.success) {
    return {
      success: false,
      status: "😢 Something went wrong while uploading your tokenURI.",
    };
  }
  const tokenURI = pinataResponse.pinataUrl;

  window.contract = await new web3.eth.Contract(contractABI, contractAddress);

  const transactionParameters = {
    to: contractAddress, // Required except during contract publications.
    from: window.ethereum.selectedAddress, // must match user's active address.
    data: window.contract.methods
      .mintNFT(window.ethereum.selectedAddress, tokenURI)
      .encodeABI(),
  };
  try {
    const txHash = await window.ethereum.request({
      method: "eth_sendTransaction",
      params: [transactionParameters],
    });
    return {
      success: true,
      status:
        "✅ Check out your transaction on Etherscan: https://sepolia.etherscan.io/tx/" +
        txHash,
    };
```

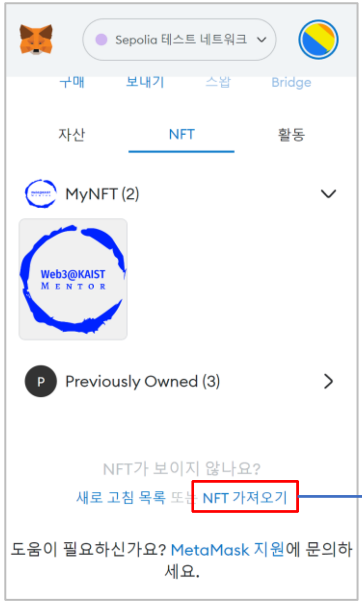
pin NFT JSON metadata and get tokenURI (IPFS CID)

create mintNFT function call tx paramaters

send tx

https://github.com/alchemyplatform/nft-minter-tutorial

Import NFTs to Metamask


Sepolia 테스트 네트워크

구매 보내기 스왑 Bridge

자산 NFT 활동

MyNFT (2)

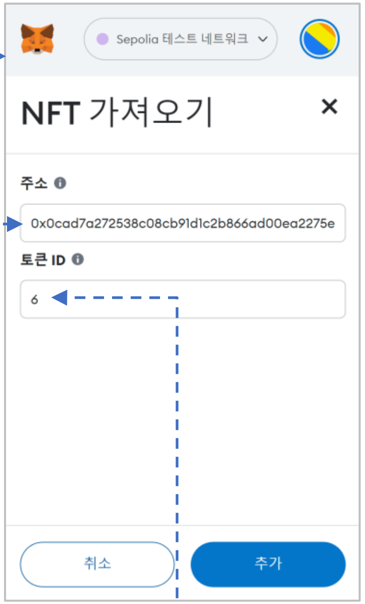
Web3@KAIST MENTOR

Previously Owned (3)

NFT가 보이지 않나요?

새로 고침 목록 또는 NFT 가져오기

도움이 필요하신가요? MetaMask 지원에 문의하세요.


Sepolia 테스트 네트워크

NFT 가져오기

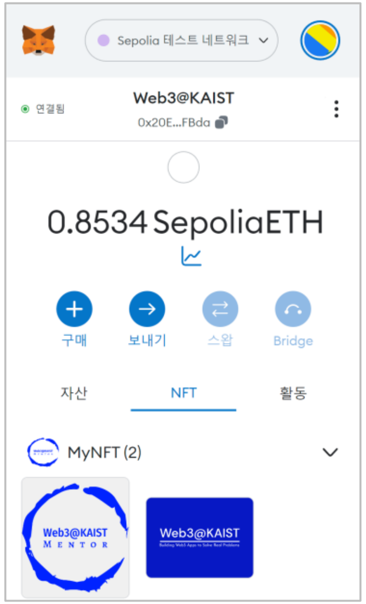
주소

0x0cad7a272538c08cb91d1c2b866ad00ea2275e

토큰 ID

6

취소 추가


Sepolia 테스트 네트워크

연결됨 Web3@KAIST 0x20E...FBda

0.8534 SepoliaETH

구매 보내기 스왑 Bridge

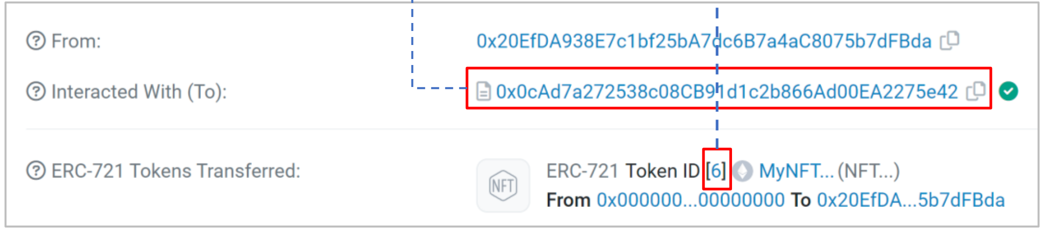
자산 NFT 활동

MyNFT (2)

Web3@KAIST MENTOR

Web3@KAIST

Etherscan


From: 0x20EfDA938E7c1bf25bA7dc6B7a4aC8075b7dFBda

Interacted With (To): 0x0cAd7a272538c08CB91d1c2b866Ad00EA2275e42

ERC-721 Tokens Transferred: ERC-721 Token ID [6] MyNFT... (NFT...)
From 0x000000...00000000 To 0x20EfDA...5b7dFBda

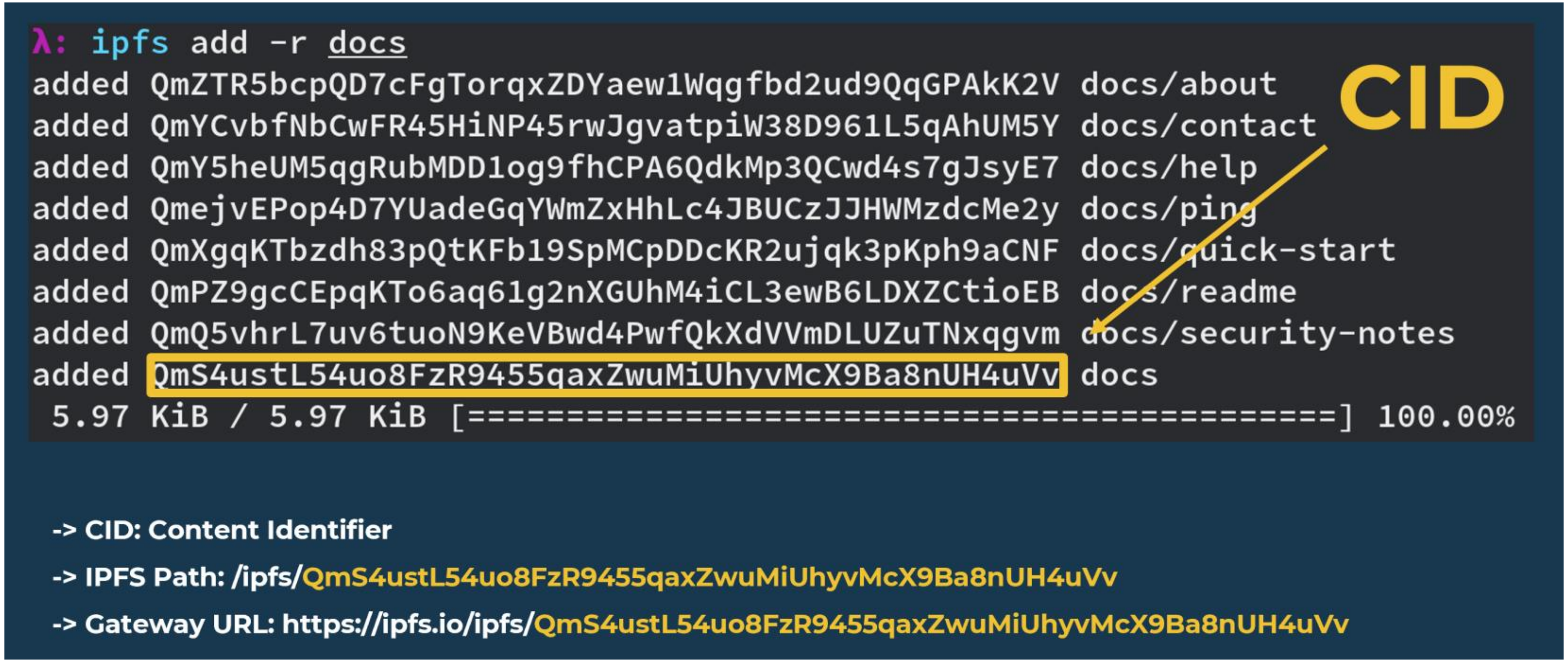
IPFS & Pinata

IPFS(InterPlanetary File System) is
a **decentralized storage and delivery network**
which builds on fundamental principles of
P2P networking and content-based addressing

IPFS Simply Explained Youtube

Addressing files in IPFS

CID = hash(file content)



```
λ: ipfs add -r docs
added QmZTR5bcpQD7cFgTorqxZDYaew1Wqgfbd2ud9QqGPAkK2V docs/about
added QmYCvbfNbCwFR45HiNP45rwJgvatpiW38D961L5qAhUM5Y docs/contact
added QmY5heUM5qgRubMDD1og9fhCPA6QdkMp3QCwd4s7gJsyE7 docs/help
added QmejvEPop4D7YUadeGqYWmZxHhLc4JBUCzJJHWMzdcMe2y docs/ping
added QmXgqKTbzdh83pQtKFb19SpMCpDDcKR2ujqk3pKph9aCNF docs/quick-start
added QmPZ9gcCEpqKTo6aq61g2nXGUhM4iCL3ewB6LDXZCtioEB docs/readme
added QmQ5vhrL7uv6tuoN9KeVBwd4PwfQkXdVVmDLUZuTNxqgvm docs/security-notes
added QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv docs
 5.97 KiB / 5.97 KiB [=================================] 100.00%
```

-> CID: Content Identifier

-> IPFS Path: /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv

-> Gateway URL: https://ipfs.io/ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv

https://docs.google.com/presentation/d/1w4n9yttx4zZSzkZgppuCsJC2jiBcW6se4XVP-hA59Yg/edit#slide=id.p1

Tamper proof in trustless nodes

Tamper proof by block hash

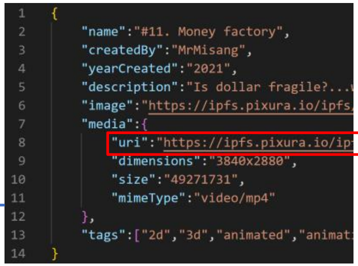
Tamper proof by content addressing

NFT Identifier

- ***ERC721 contract address***
  0xb932a70A57673d89f4acfFBE830E8ed7f75Fb9e0
- ***Token ID:*** *23418*

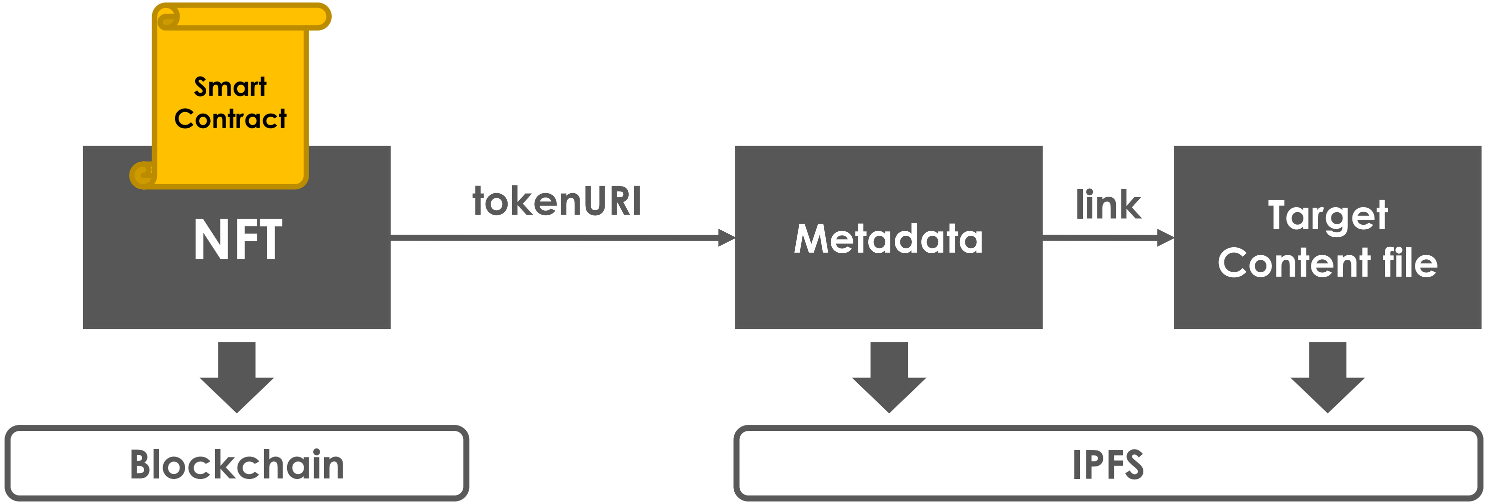
TokenURI:
ipfs://Qme5ZJugJyTajXDTRTUF42x9FajAsZYZFTBnxAWVMCy66T

Hashing

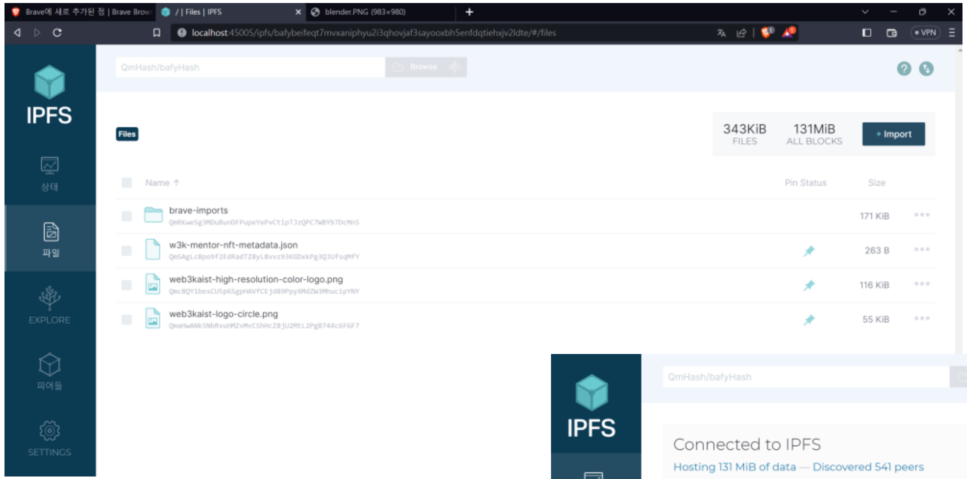


```
{
    "name":"#11. Money factory",
    "createdBy":"MrMisang",
    "yearCreated":"2021",
    "description":"Is dollar fragile?...
    "image":"https://ipfs.pixura.io/ipfs
    "media":{
        "uri":"https://ipfs.pixura.io/ip
        "dimensions":"3840x2880",
        "size":"49271731",
        "mimeType":"video/mp4"
    },
    "tags":["2d","3d","animated","animat
}
```

Hashing



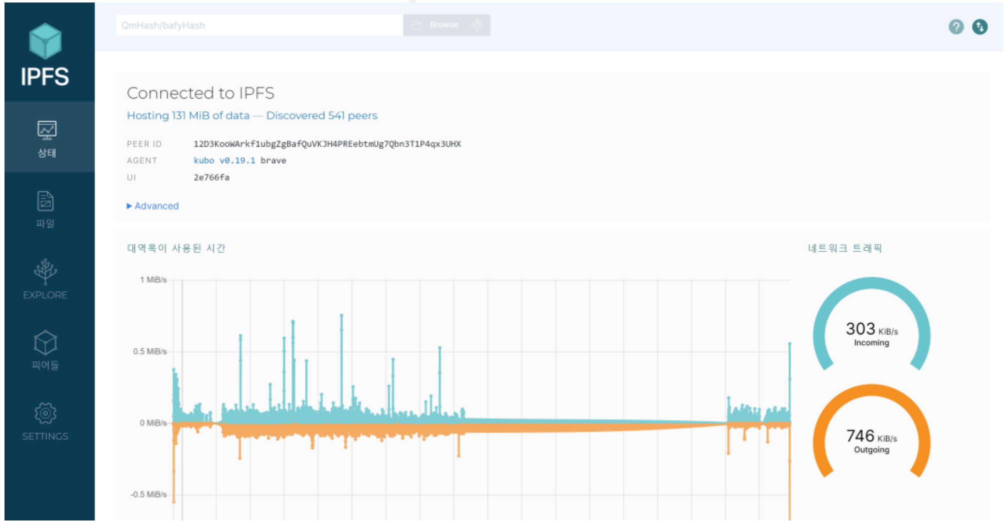
Brave browser IPFS extension


Brave에 새로 추가된 점 | Brave Brow
/ | Files | IPFS
blender.PNG (983×980)
localhost:45005/ipfs/bafybeifeqt7mvxaniphyu2i3qhovjaf3sayooxbh5enfdqtiehxjv2ldte/#/files
VPN
QmHash/bafyHash
Browse
IPFS
상태
파일
EXPLORE
피어들
SETTINGS
Files
+ Import

343KiB FILES — 131MiB ALL BLOCKS

Name ↑	Pin Status	Size
brave-imports QmRKweSg3MDuBunDFPupeYePvCt1pTJzQPC7WBYb7DcMn5		171 KiB
w3k-mentor-nft-metadata.json QmSAgLcBpo9f2EdRadTZ8yL8vvz93KGDxkPg3QJUfsqMfY	📌	263 B
web3kaist-high-resolution-color-logo.png QmcBQY1besCU5p6SgpHAVfCEjd89PpyXMdZW3MhucipYNY	📌	116 KiB
web3kaist-logo-circle.png QmaHwANk5NbRvuHMZvMvCShHcZ8jU2MtL2PgB744c6FGF7	📌	55 KiB

You can import files to IPFS and pin them



Connected to IPFS

Hosting 131 MiB of data — Discovered 541 peers

```
PEER ID    12D3KooWArkf1ubgZgBafQuVKJH4PREebtmUg7Qbn3T1P4qx3UHX
AGENT      kubo v0.19.1 brave
UI         2e766fa
```

▸ Advanced

https://ipfs.tech/#install

Availability of files in IPFS

- Importing a file in a local IPFS node
  - It returns the CID of the file, but does not mean the CID is retrievable
- **Pinning a file with the CID**
  - It allows the node to advertise that it has the CID, and provide the file to the network
- Retrieving a file from IPFS
  - It discovers the CID, fetches the file blocks and caches them
- Garbage collecting a file
  - It deletes the file blocks and frees the disk space

- Retaining a file in IPFS
  - 1) Run your IPFS node and pin the file
  - 2) **Use pinning services** that run lots of IPFS nodes and allow users to pin data on those nodes for a fee
  - Pinning services: Pinata, Web3.Storage, NFT.Storage, Infura, Filebase, etc

https://docs.ipfs.tech/concepts/lifecycle/
https://docs.ipfs.tech/concepts/persistence/#persistence-versus-permanence

IPFS gateway

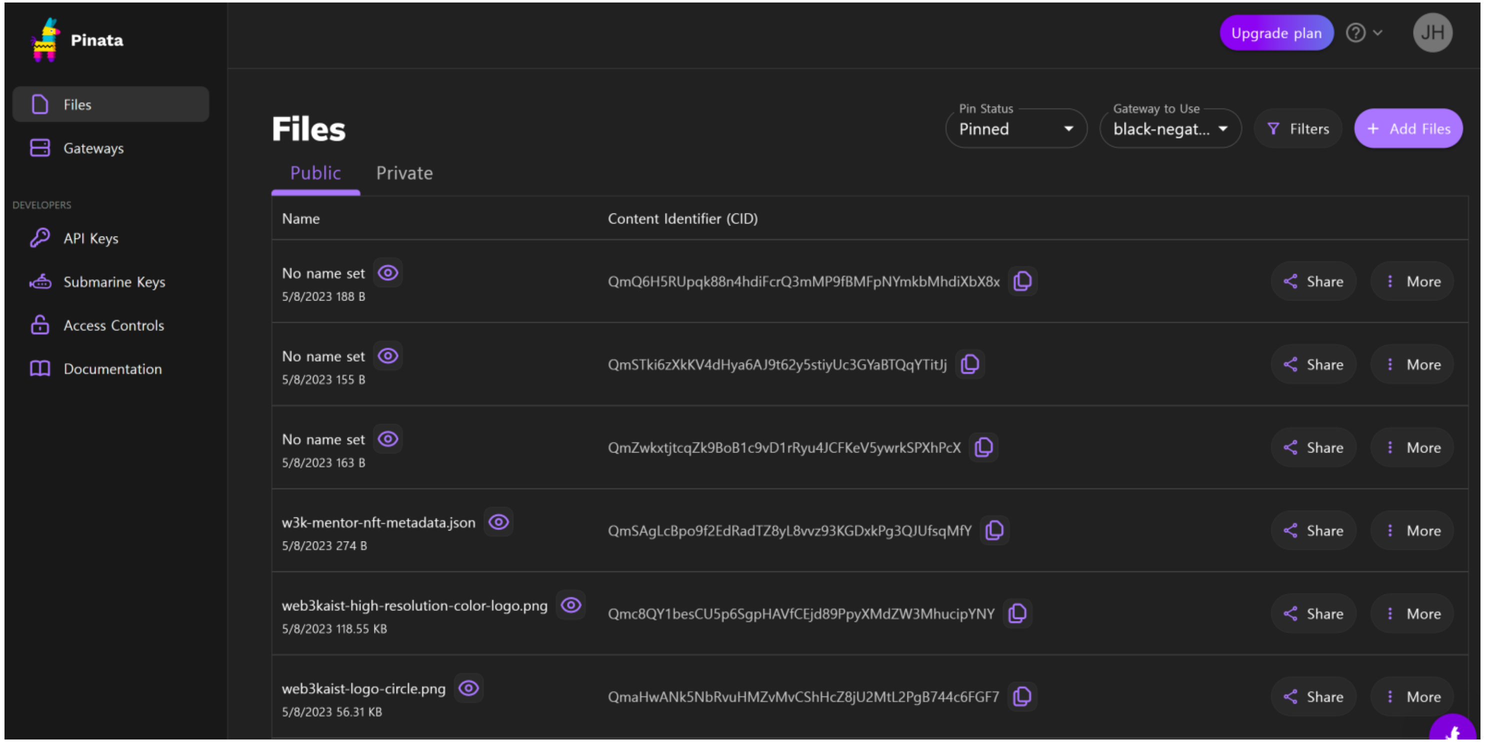
- IPFS gateway provides an HTTP-based service to access IPFS content
  IPFS path is like *ipfs://QmSAgLcBpo9f2EdRadTZ8yL8vvz93KGDxkPg3QJUfsqMfY*

- Gateway URL example
  https://ipfs.io/ipfs/QmSAgLcBpo9f2EdRadTZ8yL8vvz93KGDxkPg3QJUfsqMfY

- Gateway providers
  - Your local gateway
  - Private gateway: use cloud-based private gateway like Pinata
  - Public gateways: publicly available gateways
    - https://ipfs.io
    - https://gateway.pinata.cloud, etc
    - lists: https://ipfs.github.io/public-gateway-checker/

https://docs.ipfs.tech/concepts/ipfs-gateway/

Pinata: IPFS pinning & gateway service



Files

Pin Status: Pinned

Gateway to Use: black-negat...

Public | Private

Name	Content Identifier (CID)
No name set 5/8/2023 188 B	QmQ6H5RUpqk88n4hdiFcrQ3mMP9fBMFpNYmkbMhdiXbX8x
No name set 5/8/2023 155 B	QmSTki6zXkKV4dHya6AJ9t62y5stiyUc3GYaBTQqYTitJj
No name set 5/8/2023 163 B	QmZwkxtjtcqZk9BoB1c9vD1rRyu4JCFKeV5ywrkSPXhPcX
w3k-mentor-nft-metadata.json 5/8/2023 274 B	QmSAgLcBpo9f2EdRadTZ8yL8vvz93KGDxkPg3QJUfsqMfY
web3kaist-high-resolution-color-logo.png 5/8/2023 118.55 KB	Qmc8QY1besCU5p6SgpHAVfCEjd89PpyXMdZW3MhucipYNY
web3kaist-logo-circle.png 5/8/2023 56.31 KB	QmaHwANk5NbRvuHMZvMvCShHcZ8jU2MtL2PgB744c6FGF7

https://app.pinata.cloud/

ERC1155: Multi Token Standard

If we need multiple tokens

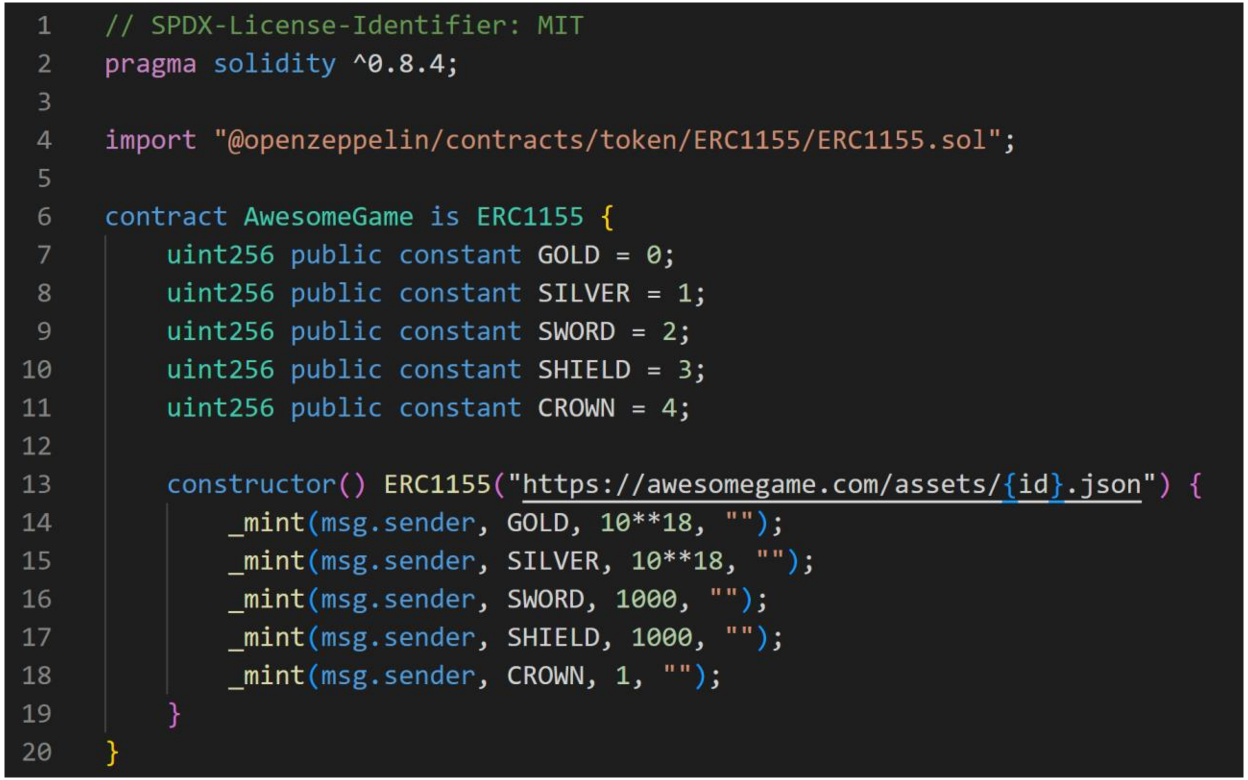
*Gold currency (10 ** 18)*	→	***ERC20 tokens***
*Silver currency (10 ** 18)*	→	***ERC20 tokens***
Sword (1,000)	→	***ERC20 tokens***
Shield (1,000)	→	***ERC20 tokens***
Crown (1)	→	***ERC721 NFTs***

Five Contracts

- High gas fee
- Redundant codes
- Managing cost

ERC1155
a single contract

ERC1155 Multi Token Example



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract AwesomeGame is ERC1155 {
    uint256 public constant GOLD = 0;
    uint256 public constant SILVER = 1;
    uint256 public constant SWORD = 2;
    uint256 public constant SHIELD = 3;
    uint256 public constant CROWN = 4;

    constructor() ERC1155("https://awesomegame.com/assets/{id}.json") {
        _mint(msg.sender, GOLD, 10**18, "");
        _mint(msg.sender, SILVER, 10**18, "");
        _mint(msg.sender, SWORD, 1000, "");
        _mint(msg.sender, SHIELD, 1000, "");
        _mint(msg.sender, CROWN, 1, "");
    }
}
```

https://docs.alchemy.com/docs/how-to-create-erc-1155-tokens

How is it possible?

All operations of blockchain such as transfer are **just changing states!**
(rewriting a ledger)

ERC1155 Multi Token Standard

- **ERC1155**: A standard interface for contracts that manage multiple token types such as fungible tokens(ERC20) and non-fungible tokens(ERC721)

Vending machine for NFTs and fungible tokens, with advanced usability features and functionality like batch transfers *- by Witek Radomski, co-creator of ERC-1155*

- **Benefits of ERC1155**

  - **Efficient Transactions**: allow for the transfer of multiple token types (fungible, non-fungible, and semi-fungible) in a single transaction, reducing tx costs and saving time

  - **Flexibility**: enable developers to create and manage various tokens for different use cases

  - **Reduced Redundancy**: reduce the redundancy on the Ethereum blockchain, conserving space and resources

  - **Safe Transfers**: provide a safe transfer function that allows tokens to be reclaimed if they are sent to the wrong address

https://blog.thirdweb.com/what-is-erc-1155-nft/

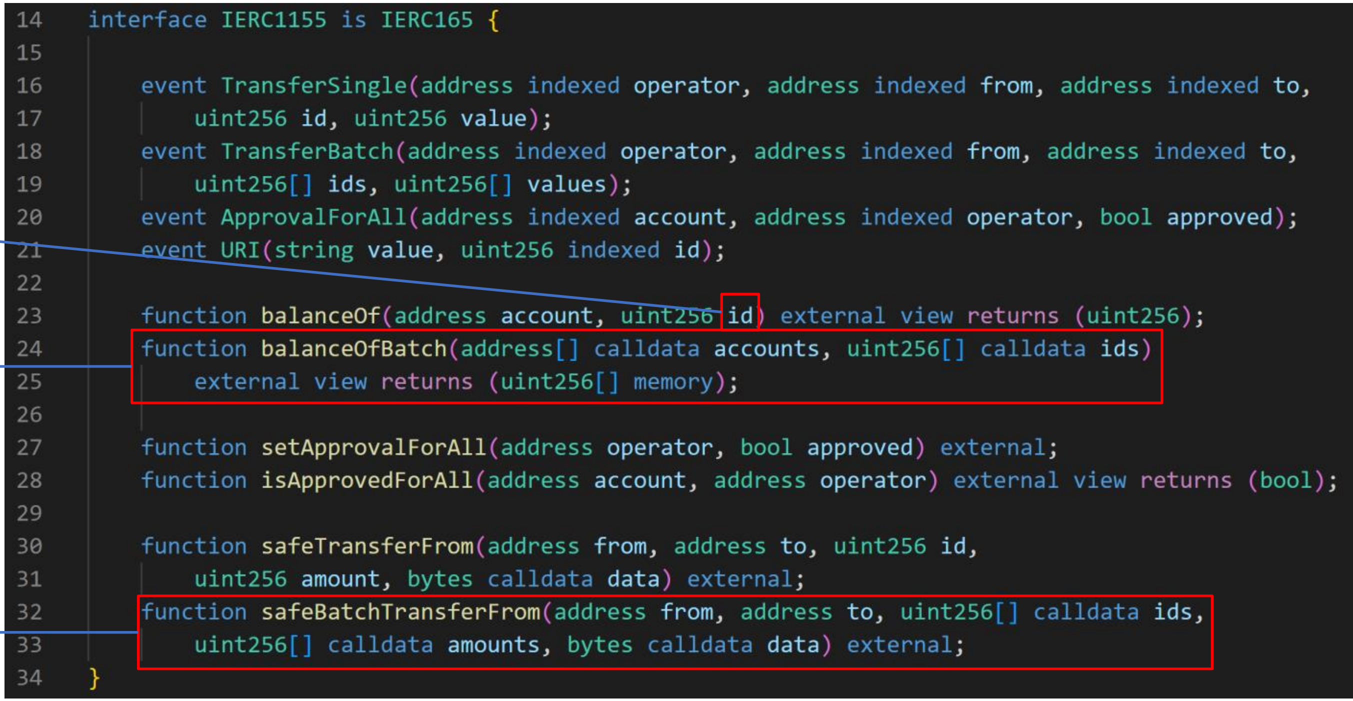
10 Projects using ERC1155

- Enjin
- Horizon Games
- OpenSea
- Rarible
- The Sandbox
- Decentraland
- Gods Unchained
- Axie Infinity
- Parallel Alpha
- SuperRare

https://blog.thirdweb.com/what-is-erc721-nft/

ERC1155 Interface

IERC1155.sol



```
interface IERC1155 is IERC165 {

    event TransferSingle(address indexed operator, address indexed from, address indexed to,
        uint256 id, uint256 value);
    event TransferBatch(address indexed operator, address indexed from, address indexed to,
        uint256[] ids, uint256[] values);
    event ApprovalForAll(address indexed account, address indexed operator, bool approved);
    event URI(string value, uint256 indexed id);

    function balanceOf(address account, uint256 id) external view returns (uint256);
    function balanceOfBatch(address[] calldata accounts, uint256[] calldata ids)
        external view returns (uint256[] memory);

    function setApprovalForAll(address operator, bool approved) external;
    function isApprovedForAll(address account, address operator) external view returns (bool);

    function safeTransferFrom(address from, address to, uint256 id,
        uint256 amount, bytes calldata data) external;
    function safeBatchTransferFrom(address from, address to, uint256[] calldata ids,
        uint256[] calldata amounts, bytes calldata data) external;
}
```

identify a token type

get the balances of multiple accounts

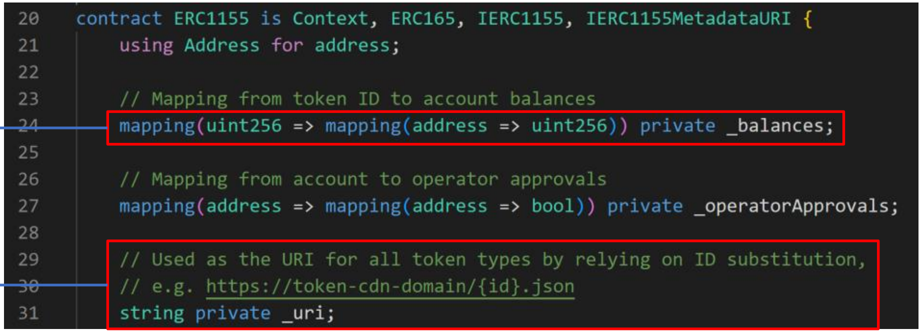
transfer multiple token types in a single tx

OpenZeppelin ERC1155 Reference plementation

ERC1155.sol

manage each token type's balances

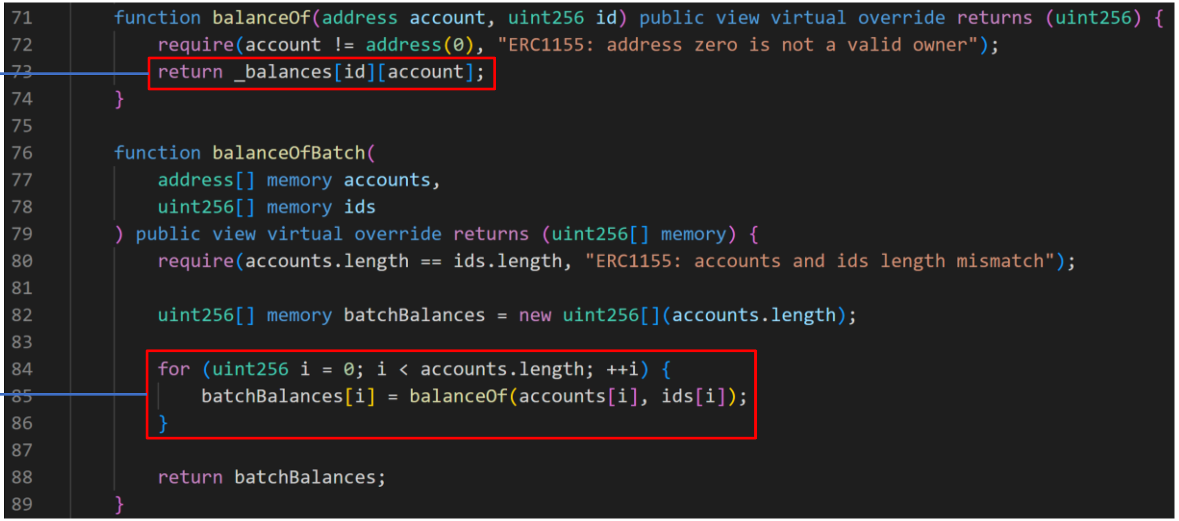
URI of Metadata JSON clients should replace {id} with actual token type ID



```
contract ERC1155 is Context, ERC165, IERC1155, IERC1155MetadataURI {
    using Address for address;

    // Mapping from token ID to account balances
    mapping(uint256 => mapping(address => uint256)) private _balances;

    // Mapping from account to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    // Used as the URI for all token types by relying on ID substitution,
    // e.g. https://token-cdn-domain/{id}.json
    string private _uri;
```

specifying token type ID

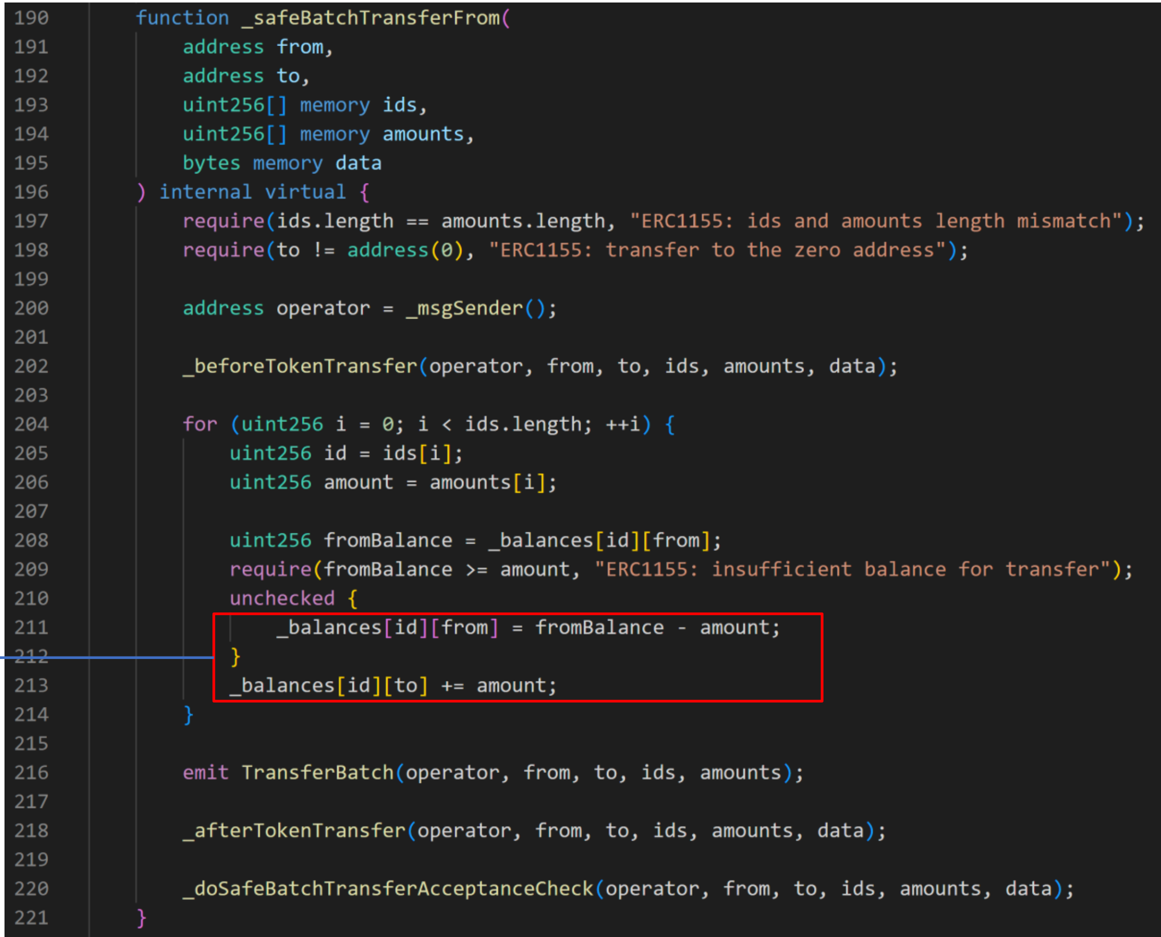
for-loop to traverse all accounts requested



```
    function balanceOf(address account, uint256 id) public view virtual override returns (uint256) {
        require(account != address(0), "ERC1155: address zero is not a valid owner");
        return _balances[id][account];
    }

    function balanceOfBatch(
        address[] memory accounts,
        uint256[] memory ids
    ) public view virtual override returns (uint256[] memory) {
        require(accounts.length == ids.length, "ERC1155: accounts and ids length mismatch");

        uint256[] memory batchBalances = new uint256[](accounts.length);

        for (uint256 i = 0; i < accounts.length; ++i) {
            batchBalances[i] = balanceOf(accounts[i], ids[i]);
        }

        return batchBalances;
    }
```

OpenZeppelin ERC1155 Reference plementation

ERC1155.sol



```
    function _safeBatchTransferFrom(
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) internal virtual {
        require(ids.length == amounts.length, "ERC1155: ids and amounts length mismatch");
        require(to != address(0), "ERC1155: transfer to the zero address");

        address operator = _msgSender();

        _beforeTokenTransfer(operator, from, to, ids, amounts, data);

        for (uint256 i = 0; i < ids.length; ++i) {
            uint256 id = ids[i];
            uint256 amount = amounts[i];

            uint256 fromBalance = _balances[id][from];
            require(fromBalance >= amount, "ERC1155: insufficient balance for transfer");
            unchecked {
                _balances[id][from] = fromBalance - amount;
            }
            _balances[id][to] += amount;
        }

        emit TransferBatch(operator, from, to, ids, amounts);

        _afterTokenTransfer(operator, from, to, ids, amounts, data);

        _doSafeBatchTransferAcceptanceCheck(operator, from, to, ids, amounts, data);
    }
```

decreasing the sender's balance and increasing the receiver's balance with a corresponding token type ID

What about NFTs?

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC1155/ERC1155.sol

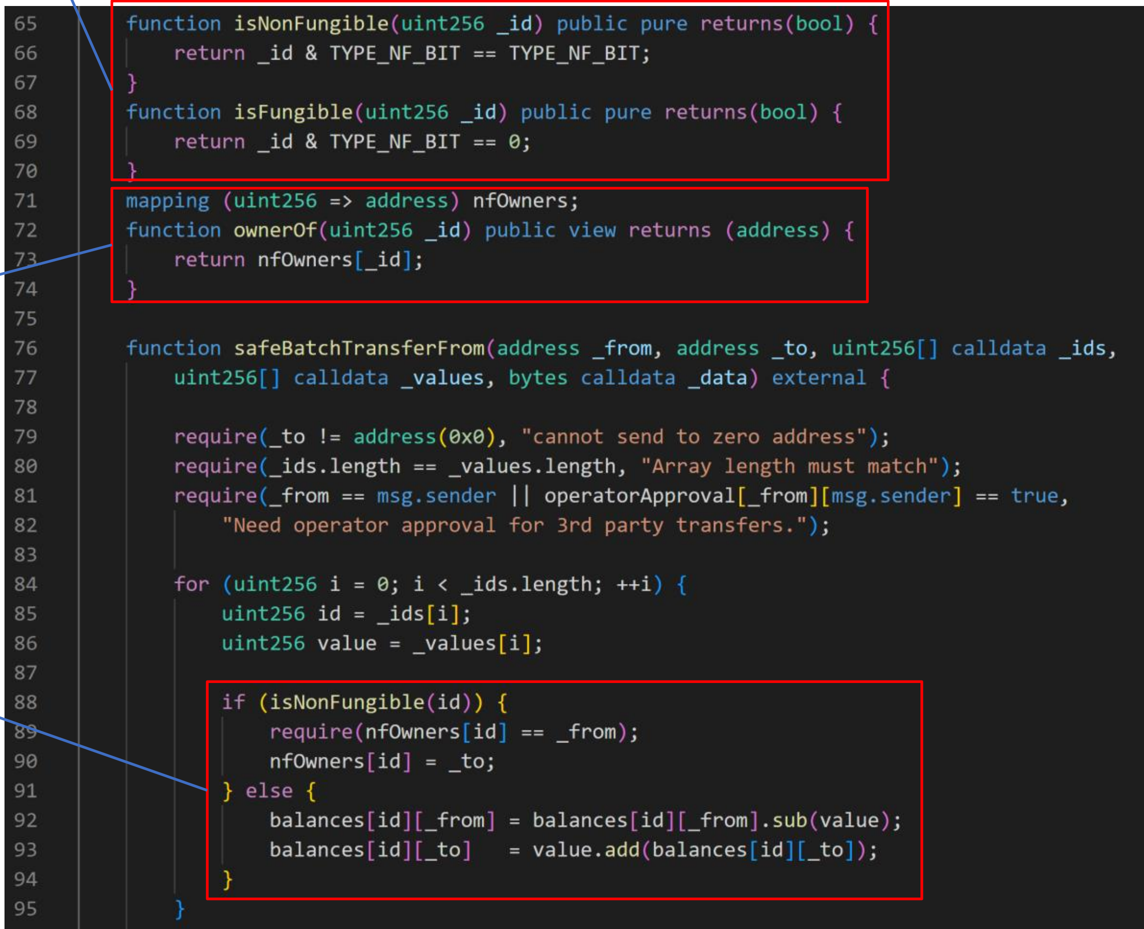
Enjin Coin ERC1155 MixedFungible Implementation

ERC1155MixedFungible.sol

determine if it's an FT or NFT with token type ID

variables and functions for NFTs

treat FTs and NFTs differently

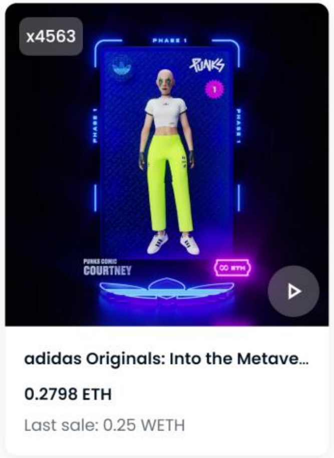


```
function isNonFungible(uint256 _id) public pure returns(bool) {
    return _id & TYPE_NF_BIT == TYPE_NF_BIT;
}
function isFungible(uint256 _id) public pure returns(bool) {
    return _id & TYPE_NF_BIT == 0;
}
mapping (uint256 => address) nfOwners;
function ownerOf(uint256 _id) public view returns (address) {
    return nfOwners[_id];
}

function safeBatchTransferFrom(address _from, address _to, uint256[] calldata _ids,
    uint256[] calldata _values, bytes calldata _data) external {

    require(_to != address(0x0), "cannot send to zero address");
    require(_ids.length == _values.length, "Array length must match");
    require(_from == msg.sender || operatorApproval[_from][msg.sender] == true,
        "Need operator approval for 3rd party transfers.");

    for (uint256 i = 0; i < _ids.length; ++i) {
        uint256 id = _ids[i];
        uint256 value = _values[i];

        if (isNonFungible(id)) {
            require(nfOwners[id] == _from);
            nfOwners[id] = _to;
        } else {
            balances[id][_from] = balances[id][_from].sub(value);
            balances[id][_to]   = value.add(balances[id][_to]);
        }
    }
```

https://github.com/enjin/erc-1155/blob/master/contracts/ERC1155MixedFungible.sol

ERC1155 Use Case: adidas Into the Metaverse NFTs

Phase 1 $\xrightarrow{\textit{redeem}}$ ***Phase 2*** $\xrightarrow{\textit{redeem}}$ ***Phase 3*** $\xrightarrow{\textit{redeem}}$ ***Phase 4***

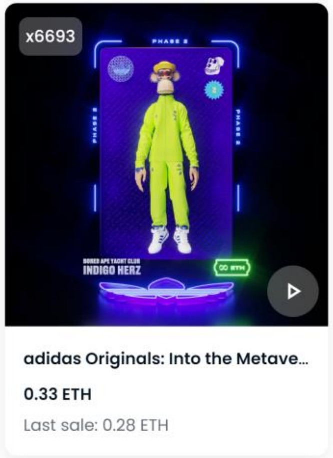


x4563

adidas Originals: Into the Metave...

0.2798 ETH

Last sale: 0.25 WETH



x6693

adidas Originals: Into the Metave...

0.33 ETH

Last sale: 0.28 ETH

???

Final NFTs

Benefits
physical goods
Virtual Gear collection

Try to understand their contract codes
https://etherscan.io/address/0x28472a58a490c5e09a238847f66a68a47cc76f0f

https://nftevening.com/adidas-metaverse-everything-you-need-to-know/
https://thenewstack.io/erc-1155-an-nft-standard-for-online-games-and-gamified-apps/

ERC5192: Soulbound Token(SBT)

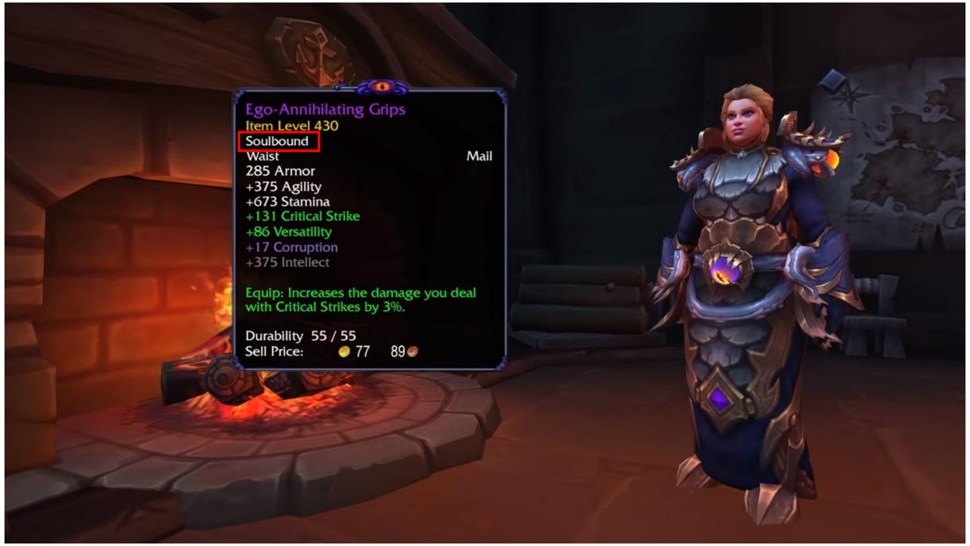
What if I send **my diploma NFT** to someone else?

Some types of NFT must be ***non-transferable***

SBT and ERC5192

- **SBT (Soulbound Token)**
  - a non-transferable NFT, which are blockchain-based digital assets that cannot be transferred to others
  - permanently tied to a specific individual, unlocking new use cases for NFTs

- **ERC5192: Minimal Soulbound NFTs**
  - Minimal interface for soulbinding EIP-721 NFTs

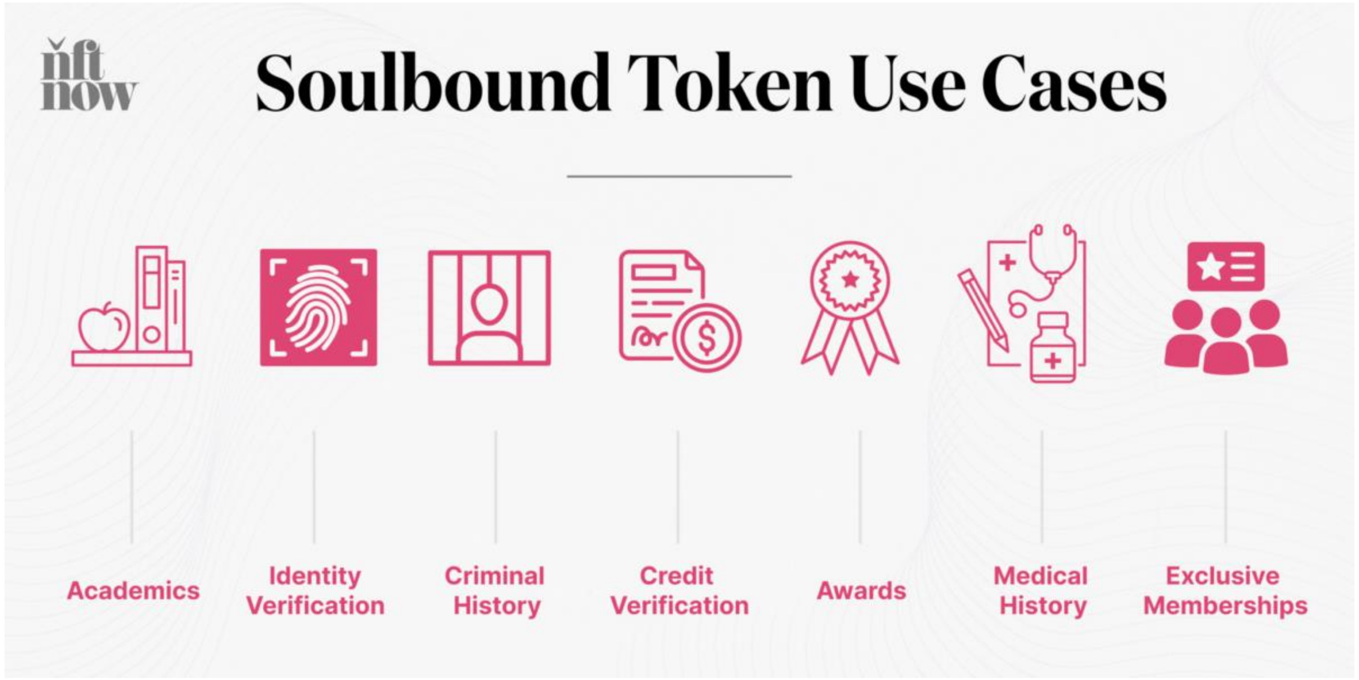


Ego-Annihilating Grips
Item Level 430
Soulbound
Waist Mail
285 Armor
+375 Agility
+673 Stamina
+131 Critical Strike
+86 Versatility
+17 Corruption
+375 Intellect

Equip: Increases the damage you deal with Critical Strikes by 3%.

Durability 55 / 55
Sell Price: 77 89

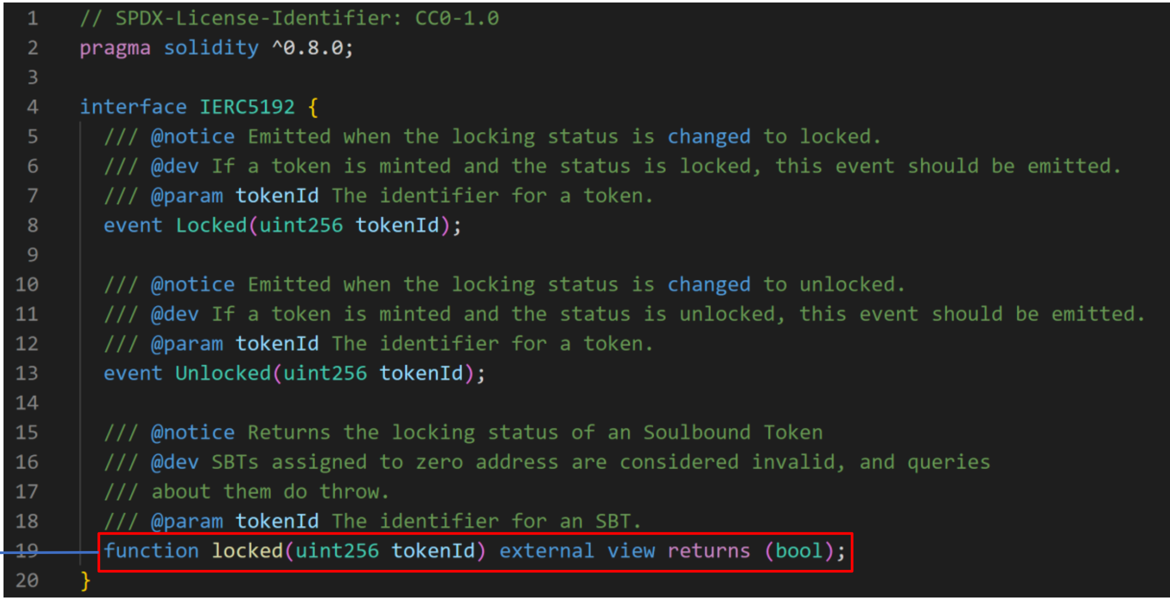
https://blog.thirdweb.com/soulbound-tokens/



https://nftnow.com/guides/soulbound-tokens-sbts-meet-the-tokens-that-may-change-your-life/
https://blog.thirdweb.com/soulbound-tokens/

ERC5192 Minimal SBT Interface

IERC5192.sol



```
// SPDX-License-Identifier: CC0-1.0
pragma solidity ^0.8.0;

interface IERC5192 {
  /// @notice Emitted when the locking status is changed to locked.
  /// @dev If a token is minted and the status is locked, this event should be emitted.
  /// @param tokenId The identifier for a token.
  event Locked(uint256 tokenId);

  /// @notice Emitted when the locking status is changed to unlocked.
  /// @dev If a token is minted and the status is unlocked, this event should be emitted.
  /// @param tokenId The identifier for a token.
  event Unlocked(uint256 tokenId);

  /// @notice Returns the locking status of an Soulbound Token
  /// @dev SBTs assigned to zero address are considered invalid, and queries
  /// about them do throw.
  /// @param tokenId The identifier for an SBT.
  function locked(uint256 tokenId) external view returns (bool);
}
```

lock the token not to transfer

https://eips.ethereum.org/EIPS/eip-5192

attestate ERC5192 SBT Implementation

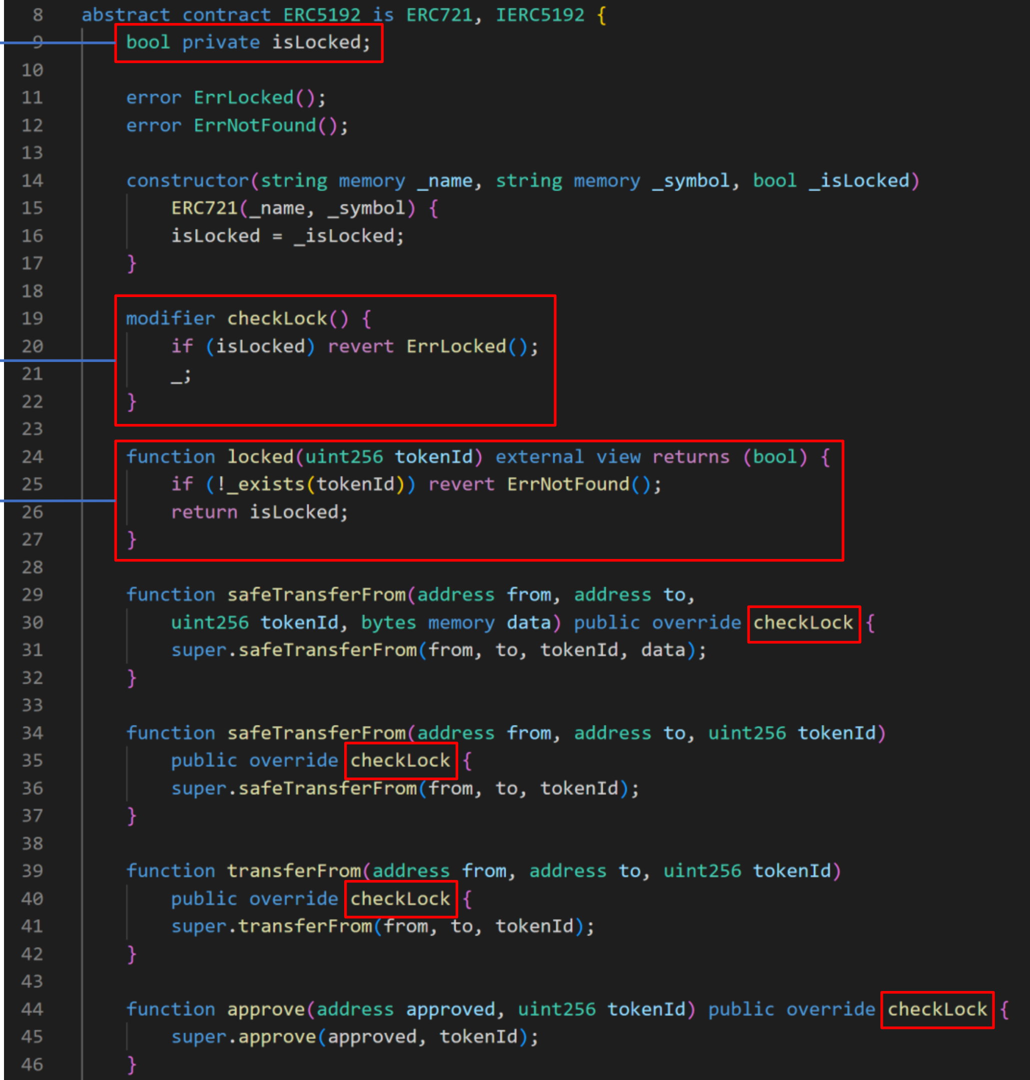
ERC5192.sol

no reference implementation of SBT from OpenZeppelin

lock check variable

Check if it's an SBT through the modifier

if this token is an SBT, the function will return true It's contract-wide



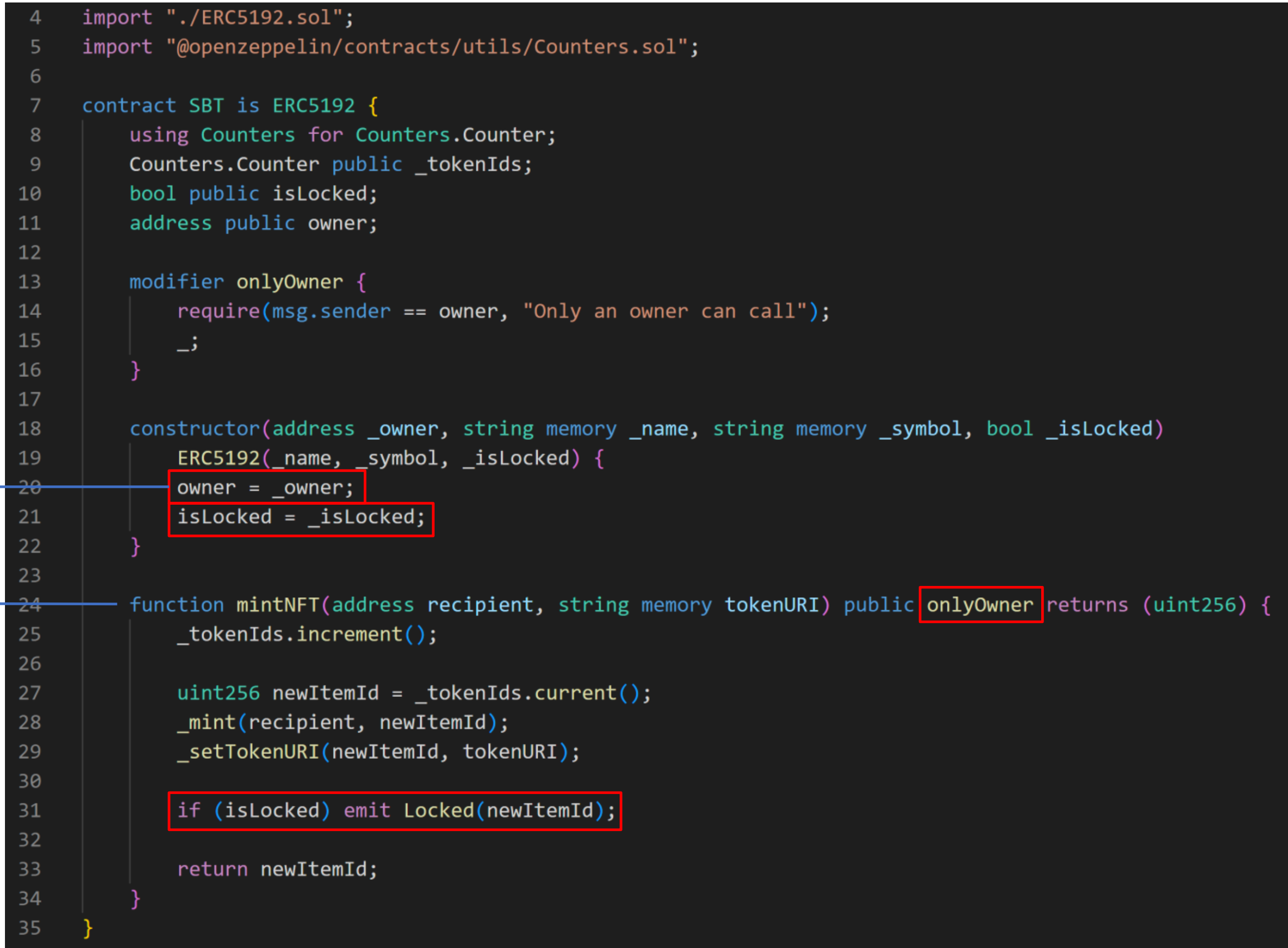
```
abstract contract ERC5192 is ERC721, IERC5192 {
    bool private isLocked;

    error ErrLocked();
    error ErrNotFound();

    constructor(string memory _name, string memory _symbol, bool _isLocked)
        ERC721(_name, _symbol) {
        isLocked = _isLocked;
    }

    modifier checkLock() {
        if (isLocked) revert ErrLocked();
        _;
    }

    function locked(uint256 tokenId) external view returns (bool) {
        if (!_exists(tokenId)) revert ErrNotFound();
        return isLocked;
    }

    function safeTransferFrom(address from, address to,
        uint256 tokenId, bytes memory data) public override checkLock {
        super.safeTransferFrom(from, to, tokenId, data);
    }

    function safeTransferFrom(address from, address to, uint256 tokenId)
        public override checkLock {
        super.safeTransferFrom(from, to, tokenId);
    }

    function transferFrom(address from, address to, uint256 tokenId)
        public override checkLock {
        super.transferFrom(from, to, tokenId);
    }

    function approve(address approved, uint256 tokenId) public override checkLock {
        super.approve(approved, tokenId);
    }
```

https://github.com/attestate/ERC5192/blob/main/src/ERC5192.sol

Minting
My ERC5192 SBTs

MySBTFactory.sol

It's almost similar to MyNFT.sol except isLocked variable



```solidity
import "./ERC5192.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract SBT is ERC5192 {
    using Counters for Counters.Counter;
    Counters.Counter public _tokenIds;
    bool public isLocked;
    address public owner;

    modifier onlyOwner {
        require(msg.sender == owner, "Only an owner can call");
        _;
    }

    constructor(address _owner, string memory _name, string memory _symbol, bool _isLocked)
        ERC5192(_name, _symbol, _isLocked) {
        owner = _owner;
        isLocked = _isLocked;
    }

    function mintNFT(address recipient, string memory tokenURI) public onlyOwner returns (uint256) {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, tokenURI);

        if (isLocked) emit Locked(newItemId);

        return newItemId;
    }
}
```

the calling contract(Factory) will be the owner

only the calling contract can call mintNFT()

Minting
My ERC5192 SBTs

MySBTFactory.sol

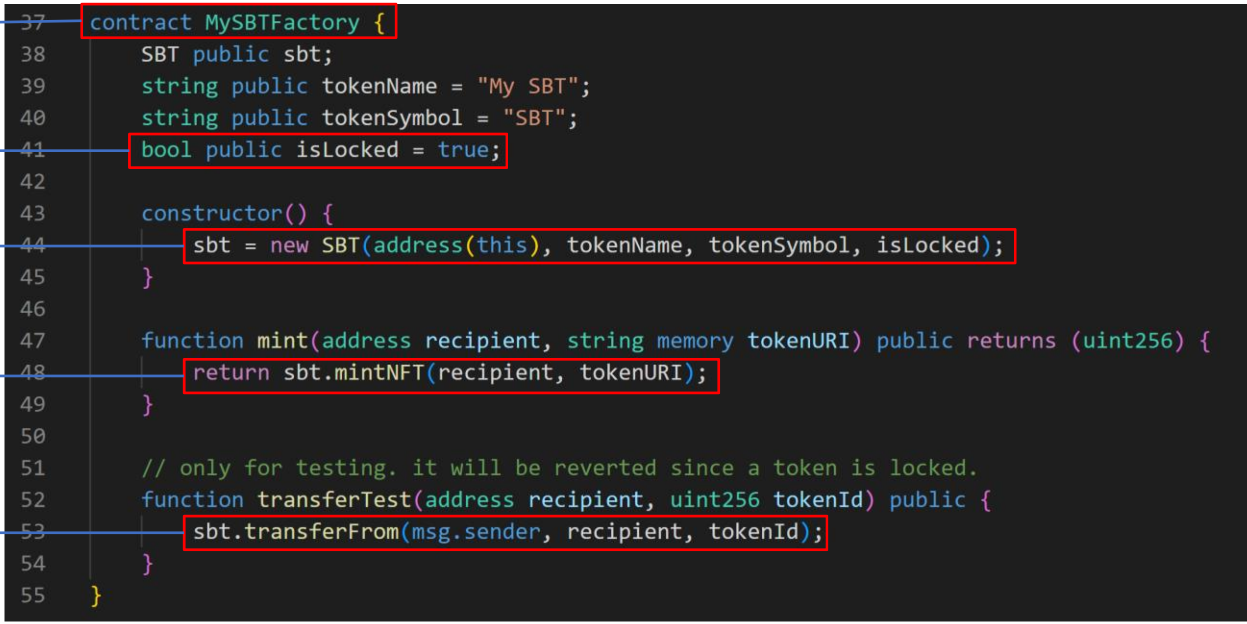
the calling contract(Factory) to create the SBT contract

if true, it runs as an SBT
if false, it runs as an NFT

create an SBT contract with setting the calling contract as the owner

mint an NFT to recipient with the tokenURI

It will be reverted then throw an error ErrLocked



```
contract MySBTFactory {
    SBT public sbt;
    string public tokenName = "My SBT";
    string public tokenSymbol = "SBT";
    bool public isLocked = true;

    constructor() {
        sbt = new SBT(address(this), tokenName, tokenSymbol, isLocked);
    }

    function mint(address recipient, string memory tokenURI) public returns (uint256) {
        return sbt.mintNFT(recipient, tokenURI);
    }

    // only for testing. it will be reverted since a token is locked.
    function transferTest(address recipient, uint256 tokenId) public {
        sbt.transferFrom(msg.sender, recipient, tokenId);
    }
}
```

Wrap-up

We Learned

- ERC721 NFT
- ERC1155 Multi Token
- ERC5129 SBT
- IPFS and Pinata