Practical session for blockchain

(9) Non-Fungible Token

Jungyoon Song[1] Giyeong Lee[1]

[1]Financial Risk Engineering Lab.
Seoul National University

May 23, 2022

Non-Fungible Token

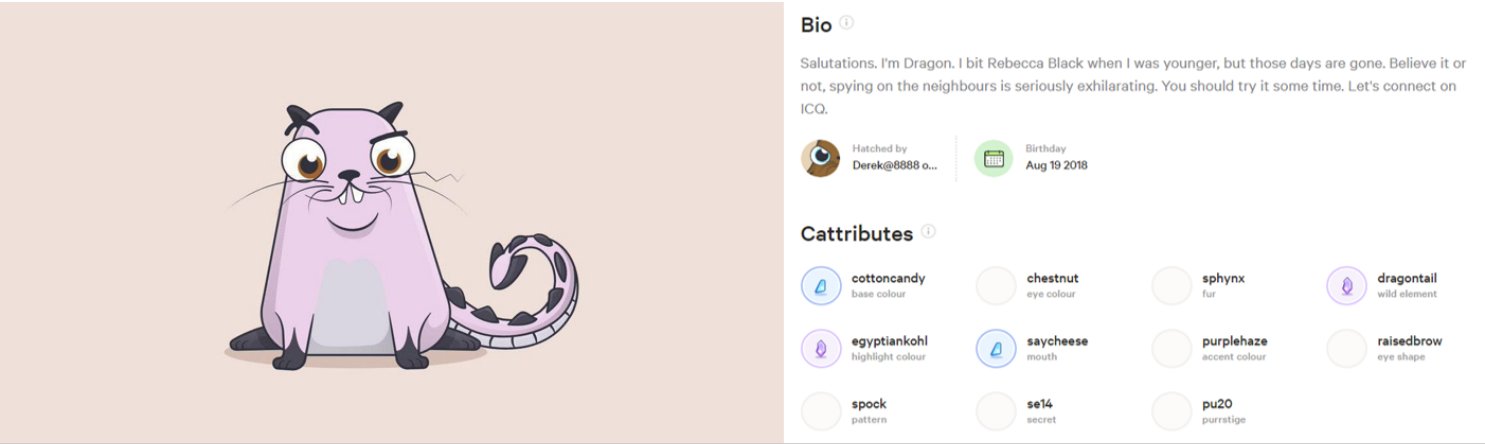
- **A Non-Fungible Token (NFT)** is used to identify something or someone in a unique way. In other words, a NFT that uses blockchain technology to prove the owner of a digital asset (Virtual Authentication Certificate).
- The **ERC-721** introduces a standard for NFT.
  (ERC-20 : Fungible Token)

Non-Fungible Token

- All NFTs have a uint256 variable called **tokenId**, so for any ERC-721 Contract, **the pair contract address, uint256 tokenId** must be globally unique.
- dApp can have a **converter** that uses the tokenId as input and outputs an image of something cool, like zombies, weapons, skills or amazing kitties
- Most NFTs trade on **OpenSea** (the largest NFT marketplace)

Examples of NFT

- **CryptoKitties** is the world's first online game based on blockchain technology and cryptocurrency Ethereum.

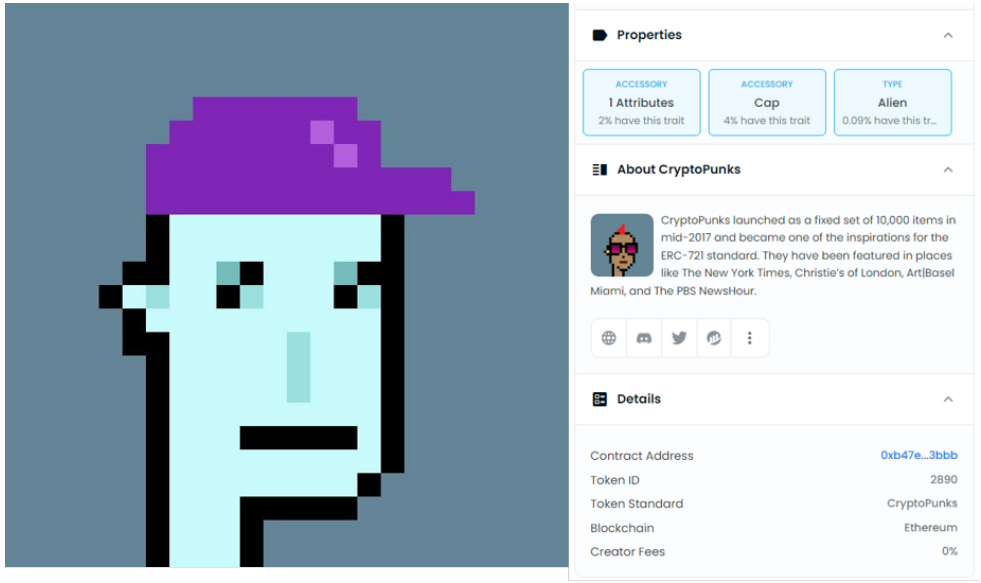


CryptoKitties #896775

Figure: *Most Expensive CryptoKitties : Dragon (Sold for 600 ETH)*

Examples of NFT

- **CryptoPunks** are developed as an applied version of the ERC-21 protocol, resulting in **the birth of the ERC-721 protocol**.
- There are 10,000 unique **CryptoPunks**, all of which are made digitally scarce through the use of blockchain technology.



Cryptopunk #2890

Figure: *Most Expensive CryptoPunks : Alien (Sold for 650 ETH)*

Examples of NFT

- **Otherdeed NFT Token**, the key to claiming land in Otherside, is the largest market capitalization based on OpenSea market.
- Recent NFT-721 protocol requires that the image URL corresponding to the TokenID be stored

Otherdeed for Otherside #33

Figure: *Most Expensive Otherdeed 63 (Sold for 333 ETH)*

Interfaces & Abstract Contracts

Interface & Abstract contract

Interfaces are similar to abstract contracts, but they cannot have any functions implemented.

- They cannot inherit from other contracts, but they can inherit from other interfaces.
- All declared functions must be external in the interface, even if they are public in the contract.
- They cannot declare a constructor.
- They cannot declare state variables.
- They cannot declare modifiers.

Abstract contracts are used as base contracts so that the child contract can inherit and utilize its functions.

[Ex. 1] Interface & Abstract contract Example

```
pragma solidity >= 0.7.0 < 0.8.0;
interface ICalculator {
  function getResult(uint, uint) external view returns (uint);
  function getResult() external view returns (bool);

}
abstract contract Test1 is ICalculator {
  //override : overwrite the function
  function getResult(uint a, uint b) external view override returns (uint) {
    uint result = a + b;
    return result;
  }
  function getResult() external view override returns (bool){
    return false;
  }
}
contract Test2 is Test1 {
  constructor() public {}
}
```

ERC-721 Interfaces & Abstract Contracts

type(I).interfaceId

A bytes4 value containing the **EIP-165** interface identifier of the given interface I. This identifier is defined as the XOR of all function selectors defined within the interface itself - excluding all inherited functions.

[Ex. 2] ERC-721 Interface (for compatibility)

```
interface IERC165 {
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}
abstract contract ERC165 is IERC165 {
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IERC165).interfaceId;
    }
}

 /* type(I).interfaceID : I 인터페이스에 내포된 모든 함수 + 인풋타입을 해쉬화하고 이 중 bytes4만으로 XOR

 Example supportsInterface  :
    bytes4 constant InterfaceSignature_ERC721Metadata = 0x5b5e139f;

    bytes4(keccak256('name()')) ^
    bytes4(keccak256('symbol()')) ^
    bytes4(keccak256('tokenURI(uint256)'));
*/
```

ERC-721 Interfaces & Abstract Contracts

[Ex. 2] ERC-721 Interface (for compatibility)

```
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);
    function balanceOf(address owner) external view returns (uint256 balance);
    function ownerOf(uint256 tokenId) external view returns (address owner);
    function safeTransferFrom(address from,address to,uint256 tokenId,bytes calldata data) external;
    function safeTransferFrom(address from,address to,uint256 tokenId) external;
    function transferFrom(address from,address to,uint256 tokenId) external;
    function approve(address to, uint256 tokenId) external;
    function setApprovalForAll(address operator, bool _approved) external;
    function getApproved(uint256 tokenId) external view returns (address operator);
    function isApprovedForAll(address owner, address operator) external view returns (bool);
}
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);
}
```

Structure and Procedures of NFT Code

[Ex. 3] ERC-721 Contract

```
# https://github.com/OpenZeppelin/openzeppelin-contracts

pragma solidity ^0.8.0;
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/utils/Context.sol";
import "@openzeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/utils/introspection/ERC165.sol";

contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    // 코드를 import하여 address와 uint256을 확장하여 사용, _msgSender()도 msg.sender 대신 사용
    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;
```

[Ex. 3] ERC-721 Contract (IERC165 and IERC721Metadata)

```
// Extract only important parts
// ERC-721 코드에서 해당 contract의 IERC721, IERC721Metadata 및 상속받은 계약에서 Interface 여부 확인
function supportsInterface(bytes4 interfaceId) public view virtual override(ERC165, IERC165)
    returns (bool) {
    return
        interfaceId == type(IERC721).interfaceId ||
        interfaceId == type(IERC721Metadata).interfaceId ||
        super.supportsInterface(interfaceId);
}

// The keyword `virtual` means that the function can change
// If you want the function to override, you need to use the `override` keyword.
// 기본적으로 해당 함수를 덮어쓰지 않으면, baseURI에 tokenID를 더한 값으로 URI 설정
// abi.encodePacked : 주어진 인수를 패킹하여 인코딩 / string(bytes) : bytes를 string으로 형변환
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
    string memory baseURI = _baseURI();
    return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}
function _baseURI() internal view virtual returns (string memory) {
    return "";
}
```

[Ex. 3] ERC-721 Contract (IERC721)

```
function balanceOf(address owner) public view virtual override returns (uint256) {
    require(owner != address(0), "ERC721: address zero is not a valid owner");
    return _balances[owner];
}
function ownerOf(uint256 tokenId) public view virtual override returns (address) {
    address owner = _owners[tokenId];
    require(owner != address(0), "ERC721: owner query for nonexistent token");
    return owner;
}
// openzeppelin에서 제공하는 코드와 동일한 기능을 하는 간소화 코드 첨부(원 코드는 실패 이유 출력부분도 있음)

// `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))` == 0x150b7a02;
// to 주소가 Contract라면 ERC721 토큰을 받을 수 있는 주소인지 확인
// 방법은 Interface와 마찬가지로 onERC721Received의 method가 구현되어있는지 확인
// onERC721Received가 구현되어있다면 True 반환 (IERC721Receiver.onERC721Received.selector == 0x150b7a02)
function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)
    internal returns (bool)
{
    if (!to.isContract()) {
        return true;
    }

    bytes4 retval = IERC721Receiver(to).onERC721Received(msg.sender, from, tokenId, _data);
    return (retval == _ERC721_RECEIVED);
}
```

Structure and Procedures of NFT Code

[Ex. 3] ERC-721 Contract (IERC721)

```
// 토큰을 받는 주소가 계약이라면, 호환이 가능한지 체크
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");
}

// 토큰을 받는 주소가 계약이라면, 호환이 가능한지 체크
// 데이터를 입력값으로 받아 주는 경우에도 가능하도록 두가지 형식의 함수를 설계
function safeTransferFrom(address from,address to,uint256 tokenId,bytes memory data)
         public virtual override {
    require(_isApprovedOrOwner(_msgSender(), tokenId),
    "ERC721: transfer caller is not owner nor approved");
    _safeTransfer(from, to, tokenId, data);
}
function safeTransferFrom(address from,address to,uint256 tokenId) public virtual
    override {
    safeTransferFrom(from, to, tokenId, "");
}
```

[Ex. 3] ERC-721 Contract (IERC721)

```
// ERC-721에서 생성된 토큰을 전송
function _transfer(address from,address to,uint256 tokenId) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");
    _beforeTokenTransfer(from, to, tokenId);
    // Clear approvals from the previous owner
    _approve(address(0), tokenId);
    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;
    emit Transfer(from, to, tokenId);
    _afterTokenTransfer(from, to, tokenId);
}
// 위임받은 사람이 토큰을 전송
function transferFrom(address from,address to,uint256 tokenId) public virtual override {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId),
            "ERC721: transfer caller is not owner nor approved");
    _transfer(from, to, tokenId);
}
```

[Ex. 3] ERC-721 Contract (IERC721)

```
// 토큰 ID의 권한을 to에게 주는 코드
function _approve(address to, uint256 tokenId) internal virtual {
    _tokenApprovals[tokenId] = to;
    emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
}
function approve(address to, uint256 tokenId) public virtual override {
    address owner = ERC721.ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");
    require(
        _msgSender() == owner || isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not owner nor approved for all"
    );
    _approve(to, tokenId);
}
// User의 권한을 to에게 주는 코드
function setApprovalForAll(address operator, bool approved) public virtual override {
    _setApprovalForAll(_msgSender(), operator, approved);
}
function _setApprovalForAll(address owner,address operator,bool approved) internal
    virtual {
    require(owner != operator, "ERC721: approve to caller");
    _operatorApprovals[owner][operator] = approved;
    emit ApprovalForAll(owner, operator, approved);
}
```

Structure and Procedures of NFT Code

[Ex. 3] ERC-721 Contract (IERC721)

```
// check based on tokenID
function getApproved(uint256 tokenId) public view virtual override returns (address) {
    require(_exists(tokenId), "ERC721: approved query for nonexistent token");
    return _tokenApprovals[tokenId];
}
// check based on User
function isApprovedForAll(address owner, address operator) public view virtual override
returns (bool) {
    return _operatorApprovals[owner][operator];
}
// spender가 권한을 위임받았는지 tokenID 기준(getApproved) 혹은 Owner 기준(isApprovedForAll)으로 확인
function _isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual
returns (bool) {
    require(_exists(tokenId), "ERC721: operator query for nonexistent token");
    address owner = ERC721.ownerOf(tokenId);
    return (spender == owner || isApprovedForAll(owner, spender) ||
            getApproved(tokenId) == spender);
}
```

[Ex. 3] ERC-721 Contract (IERC721)

```
    // tokenID에 해당하는 토큰을 생성하여 to에게 전송
    function _mint(address to, uint256 tokenId) internal virtual {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");
        _beforeTokenTransfer(address(0), to, tokenId);
        _balances[to] += 1;
        _owners[tokenId] = to;
        emit Transfer(address(0), to, tokenId);
        _afterTokenTransfer(address(0), to, tokenId);
    }
    // tokenID에 해당하는 토큰을 소각
    function _burn(uint256 tokenId) internal virtual {
        address owner = ERC721.ownerOf(tokenId);
        _beforeTokenTransfer(owner, address(0), tokenId);
        // Clear approvals
        _approve(address(0), tokenId);
        _balances[owner] -= 1;
        delete _owners[tokenId];
        emit Transfer(owner, address(0), tokenId);
        _afterTokenTransfer(owner, address(0), tokenId);
    }
}
```

[Ex. 3] ERC-721 Contract (ERC-721 URIStorage)

```
// Token URI를 설정 및 주소를 반환(converter)해주는 함수를 구현
// 기본적으로 baseURI + tokenID로 설정되지만, 실습 내용에서는 tokenURI를 입력하여 mint한다고 가정
abstract contract ERC721URIStorage is ERC721 {
    using Strings for uint256;
    // tokenID에 해당하는 tokenURI 저장 변수
    mapping(uint256 => string) private _tokenURIs;
    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        require(_exists(tokenId), "ERC721URIStorage: URI query for nonexistent token");
        string memory _tokenURI = _tokenURIs[tokenId];
        string memory base = _baseURI();
        if (bytes(base).length == 0) {
            return _tokenURI;
        }
        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;
    }
    function _burn(uint256 tokenId) internal virtual override {
        super._burn(tokenId);
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
}
```

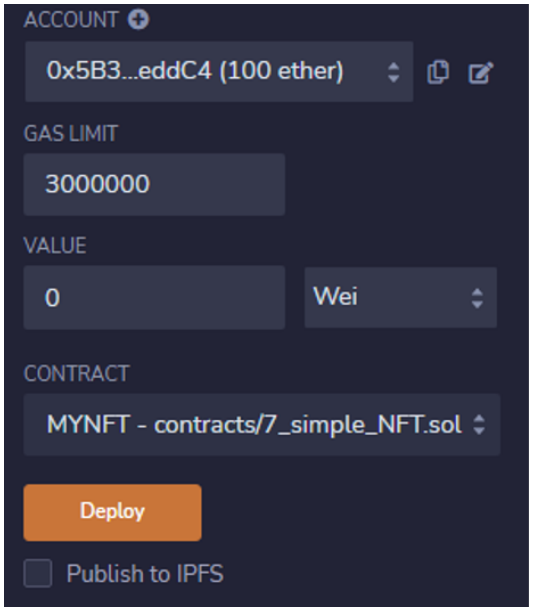
Structure and Procedures of NFT Code

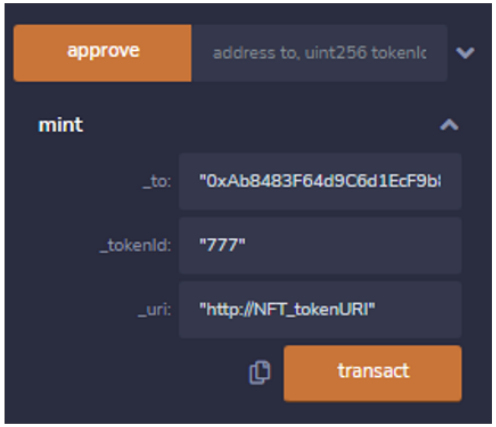
[Ex. 3] NFT Contract

```
contract MYNFT is ERC721URIStorage {
    address public nft_owner;
    constructor() ERC721("MYNFT", "nft")  {nft_owner = _msgSender();}
    modifier onlyOwner() { require(_msgSender() == nft_owner, "Only owner can run this function"); _; }
    function mint(address _to, uint256 _tokenId, string calldata _uri) onlyOwner public returns(bool){
        super._mint(_to, _tokenId);
        super._setTokenURI(_tokenId, _uri);
        return true;
  }
}
```

Implementation : NFT Contract

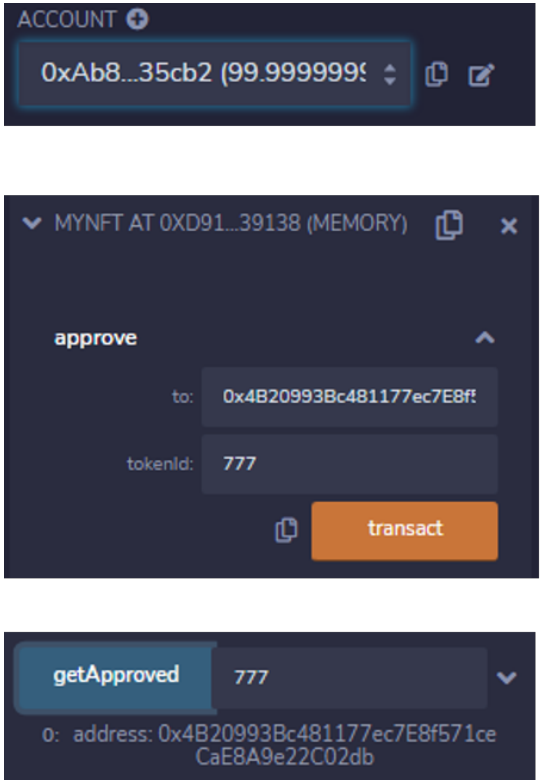
1. Deploy NFT contract and Mint NFT token

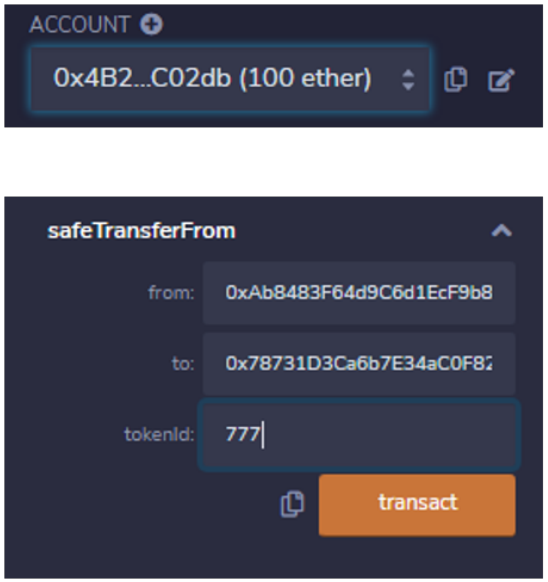


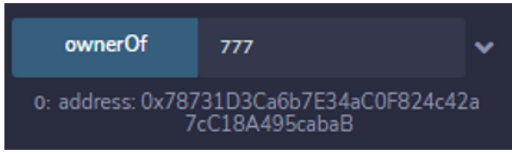


Implementation : NFT Contract

2. Delegate NFT authority to another account
3. Transfer token by authorized users







References

- https://etherscan.io/
- https://ethereum.org/en/developers/docs/standards/tokens/erc-721/
- https://guide.cryptokitties.co/guide/getting-started
- https://en.wikipedia.org/wiki/CryptoPunks
- https://docs.soliditylang.org/
- https://opensea.io/