Decentralized Application NFT Sale

Elissa Gunawan

00000027328

IF671 - Blockchain & Cryptocurrency Class A

**Penjelasan Code pada Smart Contract (Marketplace.sol)**

Import library **@openzeppelin** untuk ERC721.

import "../client/node\_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "../client/node\_modules/@openzeppelin/contracts/utils/Counters.sol";

import "../client/node\_modules/@openzeppelin/contracts/access/Ownable.sol";

import "../client/node\_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

Meng-inherit **ERC721URIStorage** dan **Ownable** agar smart contract mewarisi sifat dari standard ERC721.

contract Marketplace is ERC721URIStorage, Ownable{

Deklarasi variable untuk **productCount** untuk menghitung banyaknya produk dan **tokenId** untuk menyimpan hasil dari minting ERC721.

uint public productCount = 0;

uint public tokenId;

Variable **mapping** yang memasang uint atau integer dari product sebagai dengan struct product sebagai value.

mapping(uint => Product) public products;

**Constructor** merupakan fungsi yang hanya dijalankan sekali pada saat smart contract di deploy dimana di sini mendefinisikan nama sebagai NFTMarket dan simbol dari NFT sebagai Elissa.

constructor() ERC721("NFTMarket", "Elissa"){}

Struct **Product** merupakan struktur data yang menyimpan data-data product, yaitu id, name, description, imageurl, tokenid, price, owner, dan purchased. Yang memiliki fungsi masing-masing seperti id untuk id product, name untuk nama produk, description untuk deskripsi produk, imageurl untuk link NFT, tokenid untuk menyimpan hasil minting, price untuk harga produk, owner untuk address pemilik produk, purchased untuk true atau false mengenai barang sudah terjual atau belum.

struct Product {

        uint id;

        string name;

        string description;

        string imageurl;

        uint tokenid;

        uint price;

        address payable owner;

        bool purchased;

    }

Event merupakan suatu kejadian yang terjadi pada smart contract dan smart contract tersebut ingin berkomunikasi dengan dApps. Dimana kejadian tersebut adalah pada saat produk dibuat (**ProductCreated**) dan produk terjual (**ProductPurchased**).

event ProductCreated(

        uint id,

        string name,

        string description,

        string imageurl,

        uint tokenid,

        uint price,

        address payable owner,

        bool purchased

    );

    event ProductPurchased(

        uint id,

        string name,

        string description,

        string imageurl,

        uint tokenid,

        uint price,

        address payable owner,

        bool purchased

    );

Fungsi **mintNFT** merupakan fungsi untuk melakukan minting NFT dimana menerima 2 argumen yaitu ‘recepient’ sebagai address yang akan menerima NFT dan ‘tokenURI’ sebagai link dari NFT yang akan di-mint. Setiap fungsi mintNFT dipanggil maka variable ‘tokenIds’ akan terus bertambah sehingga id dari setiap NFT yang dibuat akan selalu unik.

function mintNFT(address recipient, string memory tokenURI) public returns (uint256){

        \_tokenIds.increment();

        uint256 newItemId = \_tokenIds.current();

        \_mint(recipient, newItemId);

        \_setTokenURI(newItemId, tokenURI);

        return newItemId;

    }

Fungsi **createProduct** merupakan fungsi untuk membuat produk NFT yang menerima beberapa argumen yaitu \_name untuk nama produk, \_description untuk deskripsi produk, \_imageurl untuk link NFT yang ingin dijual, \_price untuk harga produk. Untuk menjalankan fungsi ini ada 2 persyaratan yaitu \_name tidak boleh kosong atau panjangnya harus lebih dari 0 dan \_price tidak boleh kosong atau lebih dari 0. Setiap produk dibuat, maka productCount akan terus bertambah. Lalu dilakukan mintNFT untuk \_imageurl dan disimpan dalam tokenId, kemudian membuat produknya dan diletakkan dalam struct yang sudah dibuat sebelumnya serta memberitahukan bahwa terjadi sesuatu dalam fungsi createProduct atau memberitahukan bahwa ada produk NFT baru dengan menggunakan event ProductCreated.

function createProduct(string memory \_name, string memory \_description, string memory \_imageurl, uint \_price) public {

        require(bytes(\_name).length > 0);

        require(\_price > 0);

        productCount ++;

        tokenId = mintNFT(msg.sender, \_imageurl);

        products[productCount] = Product(productCount, \_name, \_description, \_imageurl, tokenId, \_price, payable(msg.sender), false);

        emit ProductCreated(productCount, \_name, \_description, \_imageurl, tokenId, \_price, payable(msg.sender), false);

    }

Fungsi **purchasedProduct** merupakan fungsi untuk membeli produk NFT yang hanya menerima 1 argumen yaitu id dimana id tersebut merupakan id produk. \_seller merupakan address dari owner yang memiliki produk tersebut. Persyaratan menjalankan fungsi ini adalah id produknya lebih dari 0 dan produk id tidak melebihi dari productCount atau sama dengan productCount, pastikan memiliki uang yang cukup untuk membeli, bukan merupakan produk yang sudah terjual, dan pembelinya bukan owner produk tersebut. Dan jika sudah terbeli maka owner produknya akan berubah menjadi address pembeli lalu boolean purchased akan menjadi true dan barang tersebut berubah kepemilikian yang semula dari penjual menjadi milik pembeli, lalu pembeli akan mengirimkan uangnya kepada penjual. Pada saat semua itu terjadi event ProductPurchased memberitahukan bahwa telah terjadi sesuatu dalam fungsi purchasedProduct atau memberitahukan bahwa seseorang membeli produk NFT.

function purchaseProduct(uint \_id) public payable {

        Product memory \_product = products[\_id];

        address payable \_seller = \_product.owner;

        require(\_product.id > 0 && \_product.id <= productCount);

        require(msg.value >= \_product.price);

        require(!\_product.purchased);

        require(\_seller != msg.sender);

        \_product.owner = payable(msg.sender);

        \_product.purchased = true;

        products[\_id] = \_product;

        \_transfer(ownerOf(\_product.tokenid), msg.sender, \_product.tokenid);

        payable (address(\_seller)).transfer(msg.value);

        emit ProductPurchased(productCount, \_product.name, \_product.description, \_product.imageurl, \_product.tokenid, \_product.price, payable(msg.sender), true);

    }

Cara kerja dan alur kerja website

Requirement :

* Git
* NodeJs
* Npm
* Truffle
* Ganache

Pergi ke project folder dan jalankan :

$ cd smart-contract

# install truffle

$ npm install -g truffle or npm install -g [truffle@5.4.22](mailto:truffle@5.4.22)

# download ganache

Download ganache (https://trufflesuite.com/ganache/)

# enter folder client

$ cd client

# install dependecies

$ npm install react-router-dom react-bootstrap --save

$ npm install bootstrap --save

$ npm install @openzeppelin/contracts --save

# run truffle

$ truffle compile

$ truffle migrate

# run the website

$ npm start

Buka metamask

Tambah jaringan dengan nama jaringan bebas, rpc url <http://127.0.0.1:7545>, chain id 1337, currency symbol ETH

Buka metamask

Klik akun

Pilih impor akun

Masukkin private key dari ganache