Lampiran 1. Source Code *Backend(Smart Contract)*

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.4.22 ^0.8.9;

contract Marketplace {

  string public name;

  uint public productCount = 0;

  mapping(uint => Product) public products;

  struct Product {

    uint id;

    string name;

    string desc;

    uint price;

    uint idr_price;

    uint ratio_price;

    address seller;

    address owner;

    bool purchased;

    uint256 timestamp;

  }


  event ProductCreated(

    uint id,

    string name,

    string desc,

    uint price,

    uint idr_price,

    uint256 ratio_price,
```

```
        address seller,
        address owner,
        bool purchased,
        uint256 timestamp
    );

    event ProductPurchased(
        uint id,
        string name,
        string desc,
        uint price,
        uint idr_price,
        uint ratio_price,
        address seller,
        address owner,
        bool purchased,
        uint256 timestamp
    );

    function createProduct(string memory _name, string memory _desc, uint _price, uint
_idrPrice, uint _ratio) public {
        // Require a valid name
        require(bytes(_name).length > 0);
        // Require a valid price
        require(_price > 0);
        // Increment product count
        productCount ++;
        // Create the product
        products[productCount] = Product(productCount, _name, _desc, _price, _idrPrice,
_ratio, msg.sender, msg.sender, false, block.timestamp);
```

```
    // Trigger an event
    emit ProductCreated(productCount, _name, _desc, _price, _idrPrice, _ratio,
msg.sender, msg.sender, false, block.timestamp);
  }

  function purchaseProduct(uint _id) public payable {
    // Fetch the product
    Product memory _product = products[_id];
    // Fetch the owner
    address payable _seller = payable(_product.owner);
    // Make sure the product has a valid id
    require(_product.id > 0 && _product.id <= productCount);
    // Require that there is enough Ether in the transaction
    require(msg.value >= _product.price);
    // Require that the product has not been purchased already
    require(!_product.purchased);
    // Require that the buyer is not the seller
    require(_seller != msg.sender);
    // Transfer ownership to the buyer
    _product.owner = msg.sender;
    // Change the timestamp to now
    _product.timestamp = block.timestamp;
    // Mark as purchased
    _product.purchased = true;
    // Update the product
    products[_id] = _product;
    // Pay the seller by sending them Ether
    payable(_seller).transfer(msg.value);
    // Trigger an event
```

```
        emit ProductPurchased(productCount, _product.name, _product.desc,
_product.price, _product.idr_price, _product.ratio_price, _seller, msg.sender, true,
block.timestamp);

    }

}
```

Lampiran 2. Source Code *Backend(initial_migration.js)*

```
const Migrations = artifacts.require("Migrations");

const Marketplace = artifacts.require("Marketplace");

const Market = artifacts.require("Market");

const NFT = artifacts.require("NFT");


module.exports = function (deployer) {

  deployer.deploy(Market);

  deployer.deploy(Marketplace);

  deployer.deploy(NFT);

  deployer.deploy(Migrations);

};
```

Lampiran 3. Source Code *Backend(truffle-config.js)*

```
require('dotenv').config()
/**
 * Use this file to configure your truffle project. It's seeded with some
 * common settings for different networks and features like migrations,
 * compilation and testing. Uncomment the ones you need or modify
 * them to suit your project as necessary.
 *
 * More information about configuration can be found at:
 *
 * trufflesuite.com/docs/advanced/configuration
 *
 * To deploy via Infura you'll need a wallet provider (like @truffle/hdwallet-provider)
 * to sign your transactions before they're sent to a remote public node. Infura accounts
 * are available for free at: infura.io/register.
 *
 * You'll also need a mnemonic - the twelve word phrase the wallet uses to generate
 * public/private key pairs. If you're publishing your code to GitHub make sure you load
this
 * phrase from a file you've .gitignored so it doesn't accidentally become public.
 *
 */

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

```
//
// const fs = require('fs');
const KOVAN_MNEMONIC = process.env.KOVAN_METAMASK_WALLET_SECRET;
const RINKEBY_MNEMONIC = process.env.RINKEBY_METAMASK_WALLET_SECRET;

const INFURA_SECRET = process.env.INFURA_API_SECRET;

module.exports = {
 /**
  * Networks define how you connect to your ethereum client and let you set the
  * defaults web3 uses to send transactions. If you don't specify one truffle
  * will spin up a development blockchain for you on port 9545 when you
  * run `develop` or `test`. You can ask a truffle command to use a specific
  * network from the command line, e.g
  *
  * $ truffle test --network <network-name>
  */

 networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache-cli, geth or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
  kovan: {
    provider: () => new HDWalletProvider(KOVAN_MNEMONIC,
`wss://kovan.infura.io/ws/v3/${process.env.KOVAN_INFURA_PROJECT_ID}`),
    network_id: "42",
    gas: 4000000,
```

```
    skipDryRun: true,
   },
   rinkeby: {
    provider: () => new HDWalletProvider(RINKEBY_MNEMONIC,
`wss://rinkeby.infura.io/ws/v3/${process.env.RINKEBY_INFURA_PROJECT_ID}`),
    network_id: "4",
    gas: 4000000,
    skipDryRun: true,
   },
   ropsten: {
    provider: () => new HDWalletProvider(RINKEBY_MNEMONIC,
`wss://ropsten.infura.io/ws/v3/${process.env.RINKEBY_INFURA_PROJECT_ID}`),
    network_id: "3",
    gas: 4000000,
    skipDryRun: true,
   },
   development: {
    host: "127.0.0.1",     // Localhost (default: none)
    port: 7545,            // Standard Ethereum port (default: none)
    network_id: "*",       // Any network (default: none)
   },
   // Another network with more advanced options...
   // advanced: {
   // port: 8777,            // Custom port
   // network_id: 1342,      // Custom network
   // gas: 8500000,          // Gas sent with each transaction (default: ~6700000)
   // gasPrice: 20000000000,  // 20 gwei (in wei) (default: 100 gwei)
   // from: <address>,       // Account to send txs from (default: accounts[0])
   // websocket: true        // Enable EventEmitter interface for web3 (default: false)
   // },
```

```
   // Useful for deploying to a public network.
   // NB: It's important to wrap the provider as a function.
   // ropsten: {
   // provider: () => new HDWalletProvider(mnemonic,
`https://ropsten.infura.io/v3/YOUR-PROJECT-ID`),
   // network_id: 3,       // Ropsten's id
   // gas: 5500000,        // Ropsten has a lower block limit than mainnet
   // confirmations: 2,  // # of confs 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 )
   // },
   // Useful for private networks
   // private: {
   // provider: () => new HDWalletProvider(mnemonic, `https://network.io`),
   // network_id: 2111,  // This network is yours, in the cloud.
   // production: true  // Treats this network as if it was a public net. (default: false)
   // }
  },

  // Set default mocha options here, use special reporters etc.
  mocha: {
   // timeout: 100000
  },

  // Configure your compilers
  compilers: {
   solc: {
     version: "0.8.11",    // Fetch exact version from solc-bin (default: truffle's version)
     // docker: true,       // Use "0.5.1" you've installed locally with docker (default: false)
```

```
    // settings: {          // See the solidity docs for advice about optimization and
evmVersion
    //  optimizer: {
    //    enabled: false,
    //    runs: 200
    // },
    //  evmVersion: "byzantium"
    // }
   }
 },


 // Truffle DB is currently disabled by default; to enable it, change enabled:
 // false to enabled: true. The default storage location can also be
 // overridden by specifying the adapter settings, as shown in the commented code
below.
 //
 // NOTE: It is not possible to migrate your contracts to truffle DB and you should
 // make a backup of your artifacts to a safe location before enabling this feature.
 //
 // After you backed up your artifacts you can utilize db by running migrate as follows:
 // $ truffle migrate --reset --compile-all
 //
 // db: {
   // enabled: false,
   // host: "127.0.0.1",
   // adapter: {
   //  name: "sqlite",
   // settings: {
   //     directory: ".db"
   // }
```

```
  // }
 // }
};
```

Lampiran 4. Source Code *Frontend(Web3Client.js)*

```
import Web3 from "web3";
import Marketplace from "contracts/Marketplace.json"

let selectedAccount;

// let erc20Contract;

// let isInitialized = false;

export const init = async () => {
        let provider = window.ethereum;

  if (typeof provider !== 'undefined') {
                provider.on('accountsChanged', () => window.location.reload())
                return provider
                .request({ method: 'eth_requestAccounts' })
                .then((accounts) => {
                        selectedAccount = accounts[0];
                        return selectedAccount;
                })
                .catch((err) => {
                        console.log(err);
                        return;
```

```
                });
    } else if (window.web3) {
                window.web3 = new Web3(window.web3.currentProvider)
        } else {
                console.log('Non-Ethereum browser detected. You should consider
trying MetaMask!')
        }

}

export const igetProductData = async () => {
        let provider = window.ethereum;
        let marketContract;
        const web3 = new Web3(provider);
        const networkId = await web3.eth.net.getId();
        let products = [];

        marketContract = new web3.eth.Contract(Marketplace.abi,
Marketplace.networks[networkId].address);
        let productCount = await marketContract.methods.productCount().call();
        // Load products
        for (var i = 1; i <= productCount; i++) {
    const product = await marketContract.methods.products(i).call();
                let millis = parseFloat(product.timestamp) * 1000;
                let productPrice = web3.utils.fromWei(product.price.toString(), "ether")
                let ratioPrice = web3.utils.fromWei(product.ratio_price.toString(),
"ether")
                let product_time = new Date(millis).toLocaleString();
                products.push({
                        id: product.id,
                        product_name: product.name,
```

```
                product_desc: product.desc,
                owner: product.owner,
                seller: product.seller,
                product_price: productPrice,
                ratio_price: ratioPrice,
                product_idr:  product.idr_price,
                status_buy:  product.purchased,
                timestamp: product_time
            });
        }
        return products;
}

export const createProduct = async (name, descProduct, price, idrPrice, ratio, seller) => {
        let provider = window.ethereum;
        let marketContract;
        const web3 = new Web3(provider);
        const networkId = await web3.eth.net.getId();

        marketContract = new web3.eth.Contract(Marketplace.abi,
Marketplace.networks[networkId].address);
        return marketContract.methods.createProduct(name, descProduct, price,
idrPrice, ratio).send({from: seller})
        .then(productData => {
                return productData
        }).catch(err => {
                return err.message;
        });
}
```

```
export const purchaseProduct = async (id, price, buyerAddr) => {
        let provider = window.ethereum;
        let marketContract;
        let priceToWei = Web3.utils.toWei(price, "ether")
        const web3 = new Web3(provider);
        const networkId = await web3.eth.net.getId();

        marketContract = new web3.eth.Contract(Marketplace.abi,
Marketplace.networks[networkId].address);
        return marketContract.methods.purchaseProduct(id).send({from: buyerAddr,
value: priceToWei})
        .then(productData => {
                return productData
        }).catch(err => {
                return err.message;
        });
}
```

Lampiran 5. User Guide

Untuk menjalankan *Decentralized Application*, berikut merupakan Langkah-langkah pengunaan *DApp*:

Langkah 1. Setelah *DApp* berhasil dijalankan, penjual atau pembeli melakukan login atau register untuk dompet *metamask.*

Langkah 2. Setelah berhasil login atau register, alamat dompet penjual atau pembeli akan tampil di pojok kanan atas *DApp*

Langkah 3. Dari sisi penjual, apabila penjual ingin menambah produk penjualannya, penjual dapat melakukan klik ke tombol "*Add Item"* dan akan muncul *Modal Form* penambahan produk. Dari sisi pembeli, pembeli dapat melakukan buy produk dengan melakukan klik pada tombol "*Buy*".

Langkah 4. Dari sisi penjual, penjual melakukan input data produk. Setelah data produk selesai diisi, klik tombol "Add" untuk memunculkan *Modal* konfirmasi *metamask*. Dari sisi pembeli, setelah melakukan klik tombol *"Buy"* akan melakukan *pop up modal* konfirmasi *metamask.*

Langkah 5. Dari sisi penjual, penjual melakukan konfirmasi dalam metamask untuk melanjutkan transaksi pembuatan produk pada *blockchain*. Penjual juga dapat melakukan penolakan terhadap transaksi pembuatan produk yang terjadi. Dari sisi pembeli, pembeli dapat membatalkan transaksi pembelian produk. Pembeli juga dapat melakukan konfirmasi terhadap pembelian produk.

Langkah 6. Setelah konfirmasi dilakukan, notifikasi transaksi sukses akan terlihat untuk penjual dalam transaksi pembuatan produk untuk dijual dan juga pembeli dalam transaksi pembelian produk. Transaction Hash ini dapat di cek dalam website https://kovan.etherscan.io sebagai bukti bahwa transaksi telah selesai dilaksanakan.

Langkah 7. Untuk melakukan cek riwayat transaksi, penjual dan pembeli dapat membuka *metamask* dan riwayat terdapat pada bagian *"Activity*"