Homework 8 - Interacting with NFT’s

Recap

DApps offer the benefits of interacting with blockchains to existing and novel services.

A multitude of ideas are being tested and trialled with the optimal model for each application being determined via trial and error.

DApps must try to improve on the UX on top of offering novel features to be successful.

Interact with a smart contract

1. Develop a front-end application that can call on a smart contract to mint an ERC721 token (NFT). Include relevant metadata in the token:
    1. Owner (pubkey)
    2. Token URI (e.g. an IPFS CID)
    3. Token ID
2. Retrieve information regarding an account’s NFT’s
    1. Display NFT metadata
3. Allow transfer of NFT
    1. Apply safe transfer
    2. Trade for zero cost?
4. Use smart contract from Homework 6
5. Use front-end code from Homework 7
6. Deploy on a test net

Recommended software

VSCode https://code.visualstudio.com/ (https://code.visualstudio.com/)
Node / NPM recommend use https://github.com/nvm-sh/nvm (https://github.com/nvm-sh/nvm)
Truffle https://www.trufflesuite.com/docs/truffle/getting-started/installation (https://www.trufflesuite.com/docs/truffle/getting-started/installation)
Ganache https://www.trufflesuite.com/ganache (https://www.trufflesuite.com/ganache)
Postman https://www.postman.com/ (https://www.postman.com/)
Browser (for contract development/debugging in Remix)

Code snippets

Simple starter ERC721

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

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

contract EncodeToken is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("EncodeToken", "ENC") {}

    function mintToken(address tokenOwner, string memory tokenURI)
        public
        returns (uint256)
    {
        _tokenIds.increment();

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

        return newItemId;
    }
}
```

Smart contract method call

```
// Library dependency
const Web3 = require('web3');

// Smart contract ABI
const nftAbi = require('./build/contracts/EncodeErc721.json');

// Initialise web3 library
const web3 = new Web3("http://127.0.0.1:8545");

// Set variable here for reuse
const ownerPub = '0x536F8222C676b6566FF34E539022De6c1c22cc06';

// Initialise contract
const encodeNft = new web3.eth.Contract(nftAbi.abi, "0xA97409103E409f93ecI

(async () => {
    // Get contract name
    const name = await encodeNft.methods
        .name().call();

    console.log(name);

    // Get balance
    const balance = await encodeNft.methods
        .balanceOf(ownerPub).call();

    // Log balance
    console.log(balance);
})();
```

Smart contract method send

```
// Library dependency
const Web3 = require('web3');

// Smart contract ABI
const nftAbi = require('./build/contracts/EncodeErc721.json');

// Initialise web3 library
const web3 = new Web3("http://127.0.0.1:8545");

// Set variable here for reuse
const ownerPub = '0x536F8222C676b6566FF34E539022De6c1c22cc06';

// Add credentials
web3.eth.accounts.wallet.add({
    privateKey: '4e05f623e5e7a057db5b80d8f3ae73f9efac3595ae02efca0df1f0e2
    address: ownerPub
});

// Initialise contract
const encodeNft = new web3.eth.Contract(nftAbi.abi, "0xA97409103E409f93ecE

(async () => {
    // Get contract name
    const mint = await encodeNft.methods
        .mintToken(ownerPub, "abc.com")
        .send({ from: ownerPub, gas: 500000 });

    // Log tx info
    console.log(mint);
})();
```