Assignment 5 - Develop a Document Contract.

300pts (in 2 parts)
Due Mar 11 - 100pts - the hello truffle part
Due Mar 25 - 200pts - the document contract

Part 1 - Install Truffle, Ganache

1. Install truffle, ganache-cli
2. Get the sample contract to work (MetaToken)
3. Get open-zeppelin installed
4. Get the Doc0.sol contract to compile, migrate, and run a non-useful test on it.

Part 2 - Develop a Document Contract

One of the common uses of the blockchain is as a proof of authenticity for external data. The way that this works is to take some set of external data, generate a hash for that data (SHA1 hash, or PGP2 signature example) and then write the hash to the chain. The outside code is reasonably straight forward - in some fashion you read a document, hash it - then you call a smart contract on chain to store the data on chain. The on chain contract provides the time and proof of authenticity.

A number of companies are using this kind of a system. There is an ISO900/ISO9002 document tracking company that is doing this. IOHK / BeefChain is using this approach for supply chain tracking.

This is often referred to as a “metta-data tracking” system.

In this homework we will implement the smart contract in Ethereum/Solidity to do the on-chain tracking. You will need to write the contract and to develop some tests that show that your contract works.

Develop the contract using truffle and solidity.

Turn in your contract and the test code.

Take a “document” signature and associate it with a user/acct - to mark the

1. Keep track of a users set of documents
2. Allow search for a document “hash” to see when signed
3. Keep validity of documents - time when signed
4. Noterize contract based on a list of valid notary agents.

Sample Code and Interface

Complete the following code:

File: Doc0.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.25 <0.9.0;

import "@openzeppelin/contracts/access/Ownable.sol";
```

```
contract Doc0 is Ownable {

        struct docData{
                string name;
                address owner;
                bytes32 infoHash;
                string userData;
        }
        mapping(address => docData[]) perUserDocs;
        mapping(bytes32 => address) docIndex; // docIndex[infoHash] = msg.sender;
        mapping(bytes32 => bool) infoSet;
        mapping(address => bool) isNotery;
        mapping(bytes32 => address) noterized;
        uint256 minPayment;

        event DocumentSet(string, bytes32 indexed, string);
        event DocumentOwner(address indexed, string, bytes32, string);
        event DocumentNoterized(string, bytes32, string, address indexed);
        event LogWithdrawal(address,  uint256);

        constructor() {
                minPayment = 1;
        }

        function setPayment ( uint256 p ) public onlyOwner {
                minPayment = p;
        }

        // TODO - add a getPayment function that is a public view.
        // function...

        function setNoterizer ( address aNotery ) public onlyOwner {
                isNotery[aNotery] = true;
        }

        function rmNoterizer ( address aNotery ) public onlyOwner {
                isNotery[aNotery] = false;
        }

        // TODO - add a isValidNoterizer function as a public view that returns true if the passed address is a vali
        // noterizer.
        // function...

        function newDocument ( string memory name, bytes32 infoHash, string memory info ) public payable returns(boo
                require(!infoSet[infoHash], "already set, already has owner.");         // Validate that this is a r
                require(msg.value >= minPayment, "insufficient payment to set data.");  // Validate that they are pa

                infoSet[infoHash] = true;       // This will be used in noterizeDocument

                // TODO: declare an in-memory docData structure.
                // TODO: set the values in the structure
                // TODO: append the structure to perUserDocs for this msg.sender
                // TODO: create in docIndex a way to get to this msg.sender so that the document can be found by doc
                // TODO: emit DocumentSet and DocumentOwner events

                return true;
        }

        function noterizeDocument ( string memory name, bytes32 infoHash, string memory info ) public returns (bool)
                require(infoSet[infoHash], "document not created set.");
                require(!isNotery[msg.sender], "not a registered notery.");
                noterized[infoHash] = msg.sender;               // Mark that this document has been noterized by a v
                emit DocumentNoterized(name, infoHash, info, msg.sender);
                return true;
        }
```

```
        // List Documents by Owner of Document
        function nDocuments() public view returns ( uint256 ) {
                return( perUserDocs[msg.sender].length );
        }
        function ownerOfDocument( bytes32 infoHash) public view returns ( address ) {
                return( docIndex[infoHash] );
        }
        function getDocName ( uint256 nth ) public view returns ( string memory ) {
                require(nth >= 0  && nth < perUserDocs[msg.sender].length, "nth out of range.");
                docData memory v;
                v = perUserDocs[msg.sender][nth];
                return ( v.name );
        }
        function getDocInfoHash ( uint256 nth ) public view returns ( bytes32 ) {
                require(nth >= 0  && nth < perUserDocs[msg.sender].length, "nth out of range.");
                docData memory v;
                v = perUserDocs[msg.sender][nth];
                return ( v.infoHash );
        }
        function getDocInfo ( uint256 nth ) public view returns ( string memory ) {
                require(nth >= 0  && nth < perUserDocs[msg.sender].length, "nth out of range.");
                docData memory v;
                v = perUserDocs[msg.sender][nth];
                return ( v.userData );
        }


        /**
         * Find out how much Eth the contract has accumulated.
     */
    function getBalance() public view onlyOwner returns(uint balance) {
        return address(this).balance;
    }

        /**
         * Transfer out the Eth to the owners account.
     */
    function withdraw(uint amount) public onlyOwner returns(bool success) {
        emit LogWithdrawal(msg.sender, amount);
        payable(msg.sender).transfer(amount);
        return true;
    }

}
```

Develop and write some tests for this code.