Lecture 17 - Hash / Attestation Example

Document Attestation Contract

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

import "zeppelin-solidity/contracts/ownership/Ownable.sol";

contract SignData is Ownable {

    address payable owner_address;
    uint256 private minPayment;

    mapping(uint256 => mapping(uint256 => bytes32)) dData;
    mapping(uint256 => mapping(uint256 => address)) dOowner;
    mapping(uint256 => mapping(uint256 => bool)) dMayChange;
    mapping(uint256 => mapping(uint256 => bool)) dExists;
    mapping(uint256 => mapping(uint256 => uint256)) dWhen;
    event DataChange(uint256 App, uint256 Name, bytes32 Value, address By);

    event ReceivedFunds(address sender, uint256 value, uint256 application, uint256 payFor);
    event Withdrawn(address to, uint256 amount);

    constructor() public {
        owner_address = msg.sender;
        minPayment = 1000;
    }

    modifier needMinPayment {
        require(msg.value >= minPayment, "Insufficient payment.  Must send more than minPayment.");
        _;
    }

    function init() public {
        minPayment = 1000;
    }

    function setMinPayment( uint256 _minPayment ) public onlyOwner {
        minPayment = _minPayment;
    }

    function getMinPayment() public onlyOwner view returns ( uint256 ) {
        return ( minPayment );
    }

    // --------------------------------------------------------------------------------------------------------

    /**
     * @dev Update an existing set of data if the data was created with permissions to be updated.
     */
    function setHash ( uint256 _app, uint256 _name, bytes32 _data ) public needMinPayment payable {
        address tmp = dOowner[_app][_name];
        bool mayChange = dMayChange[_app][_name];
        if ( tmp == msg.sender && !mayChange ) {
            revert("Data is not changable");
        }
        if ( tmp != msg.sender ) {
            revert("Not owner of data.");
        }
        bool ex = dExists[_app][_name];
        if ( !ex ) {
            revert("No data found." );
        }
        dData[_app][_name] = _data;
        dWhen[_app][_name] = now;
        emit DataChange(_app, _name, _data, msg.sender);
        emit ReceivedFunds(msg.sender, msg.value, _app, _name);
    }

    /**
     * @dev Create a new hash and save it's relevant data.  Check that this is a new set of data.
     */
    function createHash ( uint256 _app, uint256 _name, bytes32 _data, bool _mayChange ) public needMinPayment p
        if ( _name == 0 ) {
            revert("Invalid _name with value of 0");
        }
        if ( _app == 0 ) {
            revert("Invalid _app with value of 0");
        }
        if ( msg.sender == address(0) ) {
            revert("Invalid msg sender");
        }
        bool ex = dExists[_app][_name];
        if ( ex ) {
            revert("Data already exists for this app and name.");
        }
        dOowner[_app][_name] = msg.sender;
        dData[_app][_name] = _data;
        dMayChange[_app][_name] = _mayChange;
        dWhen[_app][_name] = now;
        dExists[_app][_name] = true;
        emit DataChange(_app, _name, _data, msg.sender);
        emit ReceivedFunds(msg.sender, msg.value, _app, _name);
    }

    /**
     * @dev return the data by looking up _app and _name in dData.  Return both the hash and the date when it w
     *      Return 0's if no data exits.
     */
    function getHash ( uint256 _app, uint256 _name ) public view returns ( bytes32, uint256 ) {
        bool ex = dExists[_app][_name];
        if ( !ex ) {
            return ( 0, 0 );
        }
        return ( dData[_app][_name], dWhen[_app][_name] );
    }

    // ----------------------------------------------------------------------------------------------------------

    /**
     * @dev payable fallback
     */
    function () external payable {
        emit ReceivedFunds(msg.sender, msg.value, 0, 1);
    }

    /**
     * @dev genReceiveFunds - generate a receive funds event.
     */
    function genReceivedFunds ( uint256 application, uint256 payFor ) public payable {
        emit ReceivedFunds(msg.sender, msg.value, application, payFor);
    }

    /**
```

```
     * @dev Withdraw contract value amount.
     */
    function withdraw( uint256 amount ) public onlyOwner returns(bool) {
        address(owner_address).transfer(amount);
        // owner_address.send(amount);
        emit Withdrawn(owner_address, amount);
        return true;
    }

    /**
     * @dev How much do I got?
     */
    function getBalanceContract() public view onlyOwner returns(uint256){
        return address(this).balance;
    }

    /**
     * @dev For futute to end the contract, take the value.
     */
    function kill() public onlyOwner {
        emit Withdrawn(owner_address, address(this).balance);
        selfdestruct(owner_address);
    }
}
```

and the test

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

/*
 * Ethereum client
 */
contract("SignData", function (accounts) {
    it("should Create contract and sign data", async function () {
        let sd = await SignData.deployed();

        let account0 = accounts[0];
        let account1 = accounts[1];
        let amount = 1000;
        var ok = true;

        //function createSignature ( uint256 _app, uint256 _name, bytes32 _data, bool _mayChange ) public needM
        var tx = await sd.createHash( 10, 4, "0x0213e3852b8afeb08929a0f448f2f693b0fc3ebe", true, {"value":amour
        // console.log ( tx );

        // function setData ( uint256 _app, uint256 _name, bytes32 _data ) public needMinPayment payable {
        tx = await sd.setHash( 10, 4, "0x11111111111afeb08929a0f448f2f693b0fc3ebe", {"value":amount} );
        // console.log ( "tx after setHash", tx );

        var x, hh, ww;
        x = await sd.getHash ( 10, 4 );
        hh = x[0];
        ww = x[1].toNumber();


        // console.log ( x, "hh=", hh, "ww=", ww );
        let expect = "0x11111111111afeb08929a0f448f2f693b0fc3ebe000000000000000000000000";
        assert.equal(hh, expect, "Invalid stored hash");
        if ( hh != expect ) {
            ok = false;
        }

        var today = new Date();
        var sDate = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();

        var timestamp = ww * 1000;    // Convert from number of seconds (Eth) since 1970 to mili-seconds
        var date = new Date(timestamp);
        // console.log(date.getDate())
        // console.log(date, sDate) // 2022-03-04T14:40:57.000Z
        var gDate = date.getFullYear()+'-'+(date.getMonth()+1)+'-'+date.getDate();
        assert.equal(sDate,gDate, "Invalid date for this hash.");
        if ( sDate != gDate ) {
            ok = false;
        }

        return assert.isTrue(ok);
    });
});
```