Lecture 19 - Hash / Attestation Example

Document Attestation Contract

```
 1: // SPDX-License-Identifier: MIT
 2: pragma solidity >=0.4.22 <0.9.0;
 3:
 4: import "zeppelin-solidity/contracts/ownership/Ownable.sol";
 5:
 6: contract SignData is Ownable {
 7:
 8:     address payable owner_address;
 9:     uint256 private minPayment;
10:
11:     mapping(uint256 => mapping(uint256 => bytes32)) dData;
12:     mapping(uint256 => mapping(uint256 => address)) dOowner;
13:     mapping(uint256 => mapping(uint256 => bool)) dMayChange;
14:     mapping(uint256 => mapping(uint256 => bool)) dExists;
15:     mapping(uint256 => mapping(uint256 => uint256)) dWhen;
16:     event DataChange(uint256 App, uint256 Name, bytes32 Value, address By);
17:
18:     event ReceivedFunds(address sender, uint256 value, uint256 application, uint256 payFor);
19:     event Withdrawn(address to, uint256 amount);
20:
21:     constructor() public {
22:         owner_address = msg.sender;
23:         minPayment = 1000;
24:     }
25:
26:     modifier needMinPayment {
27:         require(msg.value >= minPayment, "Insufficient payment.  Must send more than minPayment.");
28:         _;
29:     }
30:
31:     function init() public {
32:         minPayment = 1000;
33:     }
34:
35:     function setMinPayment( uint256 _minPayment ) public onlyOwner {
36:         minPayment = _minPayment;
37:     }
38:
39:     function getMinPayment() public onlyOwner view returns ( uint256 ) {
40:         return ( minPayment );
41:     }
42:
43:     // ----------------------------------------------------------------------------------------------------
44:
45:     /**
46:      * @dev Update an existing set of data if the data was created with permissions to be updated.
47:      */
48:     function setHash ( uint256 _app, uint256 _name, bytes32 _data ) public needMinPayment payable {
49:         address tmp = dOowner[_app][_name];
50:         bool mayChange = dMayChange[_app][_name];
51:         if ( tmp == msg.sender && !mayChange ) {
52:             revert("Data is not changable");
53:         }
54:         if ( tmp != msg.sender ) {
55:             revert("Not owner of data.");
```

```
56:         }
57:         bool ex = dExists[_app][_name];
58:         if ( !ex ) {
59:             revert("No data found." );
60:         }
61:         dData[_app][_name] = _data;
62:         dWhen[_app][_name] = now;
63:         emit DataChange(_app, _name, _data, msg.sender);
64:         emit ReceivedFunds(msg.sender, msg.value, _app, _name);
65:     }
66:
67:     /**
68:      * @dev Create a new hash and save it's relevant data.  Check that this is a new set of data.
69:      */
70:     function createHash ( uint256 _app, uint256 _name, bytes32 _data, bool _mayChange )
71:         public needMinPayment payable {
72:         if ( _name == 0 ) {
73:             revert("Invalid _name with value of 0");
74:         }
75:         if ( _app == 0 ) {
76:             revert("Invalid _app with value of 0");
77:         }
78:         if ( msg.sender == address(0) ) {
79:             revert("Invalid msg sender");
80:         }
81:         bool ex = dExists[_app][_name];
82:         if ( ex ) {
83:             revert("Data already exists for this app and name.");
84:         }
85:         dOowner[_app][_name] = msg.sender;
86:         dData[_app][_name] = _data;
87:         dMayChange[_app][_name] = _mayChange;
88:         dWhen[_app][_name] = now;
89:         dExists[_app][_name] = true;
90:         emit DataChange(_app, _name, _data, msg.sender);
91:         emit ReceivedFunds(msg.sender, msg.value, _app, _name);
92:     }
93:
94:     /**
95:      * @dev return the data by looking up _app and _name in dData.  Return both the hash and the date when
96:      *      it was stored..  Return 0's if no data exits.
97:      */
98:     function getHash ( uint256 _app, uint256 _name ) public view returns ( bytes32, uint256 ) {
99:         bool ex = dExists[_app][_name];
100:        if ( !ex ) {
101:            return ( 0, 0 );
102:        }
103:        return ( dData[_app][_name], dWhen[_app][_name] );
104:    }
105:
106:    // ----------------------------------------------------------------------------------------------------
107:
108:    /**
109:     * @dev payable fallback
110:     */
111:    function () external payable {
112:        emit ReceivedFunds(msg.sender, msg.value, 0, 1);
113:    }
114:
115:    /**
116:     * @dev genReceiveFunds - generate a receive funds event.
117:     */
118:    function genReceivedFunds ( uint256 application, uint256 payFor ) public payable {
119:        emit ReceivedFunds(msg.sender, msg.value, application, payFor);
120:    }
121:
```

```
122:     /**
123:      * @dev Withdraw contract value amount.
124:      */
125:     function withdraw( uint256 amount ) public onlyOwner returns(bool) {
126:         address(owner_address).transfer(amount);
127:         // owner_address.send(amount);
128:         emit Withdrawn(owner_address, amount);
129:         return true;
130:     }
131: 
132:     /**
133:      * @dev How much do I got?
134:      */
135:     function getBalanceContract() public view onlyOwner returns(uint256){
136:         return address(this).balance;
137:     }
138: 
139:     /**
140:      * @dev For fututе to end the contract, take the value.
141:      */
142:     function kill() public onlyOwner {
143:         emit Withdrawn(owner_address, address(this).balance);
144:         selfdestruct(owner_address);
145:     }
146: }
```

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 createHash ( uint256 _app, uint256 _name, bytes32 _data, bool _mayChange ) public
        // needMinPayment payable {
        var tx = await sd.createHash( 10, 4, "0x0213e3852b8afeb08929a0f448f2f693b0fc3ebe", true,
            {"value":amount} );
        // 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);
    });
});
```