Lecture 23 - web3.js client example

To setup MetaMask to work with truffle: https://trufflesuite.com/docs/truffle/getting-started/truffle-with-metamask

The Truffle PetShop Tutorial: https://trufflesuite.com/guides/pet-shop/

Steps

1. Setting up the development environment (you should be able to write, build, compile a contact at this point)
2. Writing the smart contract
3. Compiling and migrating the smart contract
4. Testing the smart contract
5. Creating a user interface to interact with the smart contract
6. Running a “server”
7. Interacting with the dApp in a browser

Setting up the development environment

Use git to clone this repository `./eth` . If you move this to a different path then `cd eth ; rm -rf build` to get rid of the old build (it has paths in it that are now wrong). Then recompile to build again.

```
$ truffle compile
```

Use truffle to pull out a pre-built environment like the pet-shop.

Remember to run ganache with the web-sockets turned on.

```
$ ganache  --server.ws
```

Check MetaMask configuration to get it to talk to gouache ganache at

```
RPC Listening on 127.0.0.1:8545
```

Save the list of accounts - we may need that later and it is unique each time you run ganache

```
Available Accounts
==================
(0) 0x31D7f4435fAcd54A4941447624f8Dd2024e0382D (1000 ETH)
(1) 0x97FCbbFa2B46d7e8f572864E917d91B9FaA53a71 (1000 ETH)
(2) 0xEF712f61EA21d683d9eA4b11f927B24cDB373F74 (1000 ETH)
(3) 0x9d934D43B20607ce1B5096D21fb7C6C1F9d50b56 (1000 ETH)
(4) 0x79668fF2E8e12570B96E342Ac1362B3a3bF6eE47 (1000 ETH)
(5) 0xa82998A94C94132a154a2Ce1a5A49F587648Db7a (1000 ETH)
(6) 0xe34fA6f30f22F87639055FEa72653Fb4785263e0 (1000 ETH)
(7) 0x27CbED781f167fCDe94BE5578F4d801c0F7E83B6 (1000 ETH)
(8) 0x1F54130FB40E97FD49c9cEF928958494Da20A030 (1000 ETH)
```

```
(9) 0x6827Db0342cd1EC78D12735dFE5898bf9685d262 (1000 ETH)

Private Keys
==================
(0) 0xb3128b28ab6963f34170ddafa2cf820e160110c0ef4f0c8b70b6e71f40199566
(1) 0x3afce7e5e265fceee780167905da8b0b9e398a93cbd978f129a504f0874dbf95
(2) 0xdcd7fb4f680e1b0220024fe381cd8685a496e59bafab894c02212e8faf7dd54f
(3) 0xd1005931c634c4758b8f64f7b706243978c12b065b0cf0c0e6d8591ec75e3404
(4) 0x6ff4eb43bb56a6b2ec91630870781c009e1d2857db7af1e1a167ca57972f04b8
(5) 0x09d2f7a43e1a55c307cf2d16decf0ddb3bfd0159c8ef6b0abd31c88aea85fb03
(6) 0x7b0768ba75b9fbc95330befd30b8a8c11fec71fee6d2bb97c13636c984f92978
(7) 0xf4f6088707b1b2376b60e545c4b6f01452a5a2c58ce7111bd3345f9db3edbb2e
(8) 0x6483311ee5caef63d8a5004ff20d459b4e1d8a3ce4b95753de04d9cf6e780e1e
(9) 0xbb0e78a03ae7db784cf544b7d72d25f71d7a8bdbc51600737f05fbc1df9ae48f
```

Remember that you have to load your contracts every time you start ganache.

```
$ truffle migrate --reset
```

Save the output from the migrate (Look at `migrate/3_saved_message.js`

```
3_saved_message.js
==================

   Replacing 'SavedMessage'
   ------------------------
   > transaction hash:    0x6f9415fed8259623c80ca000ddba00d8ea0db003aa3a8dcfb98106b928910dec
   > Blocks: 0            Seconds: 0
   > contract address:    0xD0C9933ed2f7022177E4fa572F3Ce7DFb0f2B8C7
   > block number:        85
   > block timestamp:     1647633386
   > account:             0x31D7f4435fAcd54A4941447624f8Dd2024e0382D
   > balance:             999.924376116511205725
   > gas used:            320309 (0x4e335)
   > gas price:           2.500015457 gwei
   > value sent:          0 ETH

   > total cost:          0.000800777451016213 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.000800777451016213 ETH
```

File: migrations/3_saved_message.js:

Writing the smart contract

Write your new smart contract. In this case we are going to use a Hello World contract.

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

contract SavedMessage {
    string public currentMessage = "Hello World, from Ethereum";

    constructor() {
    }

    function getCurrentMessageTxt() external view returns (string memory) {
        return currentMessage;
    }

    function setCurrentMessageTxt(string calldata _msg) external {
        currentMessage = _msg;
    }
}
```

Compiling and migrating the smart contract

Check that your migration file is correct for any new contracts. The stuff in the `./migrations` directory is run by sorted order, hence the *1*, *2* etc.

```
$ truffle migrate --reset
```

Testing the smart contract

```
$ truffle test
```

output:

```
(base) philip@victoria eth % truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


  Contract: SavedMessage
{
  tx: '0xf7e30424fcfdab148563cacbb7a5e7f9de5dca9f34c43c5cf098dd6b1b62ac30',
  receipt: {
    transactionHash: '0xf7e30424fcfdab148563cacbb7a5e7f9de5dca9f34c43c5cf098dd6b1b62ac30',
    transactionIndex: 0,
    blockNumber: 80,
    blockHash: '0x84754a3d0bf83bac612e1b8cffe6c244cbdd90ee2ff233ad346012deb73c943f',
    from: '0x31d7f4435facd54a4941447624f8dd2024e0382d',
    to: '0x1c9c41c321b49f33cbf1a7a2a3d6dd292ccefd11',
    cumulativeGasUsed: 29752,
    gasUsed: 29752,
    contractAddress: null,
    logs: [],
    logsBloom: '0x0000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    status: true,
    effectiveGasPrice: '0x95036c93',
    type: '0x2',
    rawLogs: []
  },
  logs: []
}
    ✓ should Create contract (93ms)


  1 passing (175ms)
```

You *have to copy the files from ./build/contract into ../client/contract*. This is in the Makefile.

Creating a user interface to interact with the smart contract

2 Files on the client side:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="shortcut icon" type="image/png" href="favicon.png" />
    <title>Hello World dApp</title>
<style>
.button-style {
   background: #3D94F6;
   background-image: -webkit-linear-gradient(top, #3D94F6, #1E62D0);
   background-image: -moz-linear-gradient(top, #3D94F6, #1E62D0);
   background-image: -ms-linear-gradient(top, #3D94F6, #1E62D0);
   background-image: -o-linear-gradient(top, #3D94F6, #1E62D0);
```

```
        background-image: -webkit-gradient(to bottom, #3D94F6, #1E62D0);
        -webkit-border-radius: 6px;
        -moz-border-radius: 6px;
        border-radius: 6px;
        color: #FFFFFF;
        font-family: Arial;
        font-size: 12px;
        font-weight: 300;
        padding: 11px;
        -webkit-box-shadow: 1px 1px 20px 0 #000000;
        -moz-box-shadow: 1px 1px 20px 0 #000000;
        box-shadow: 1px 1px 20px 0 #000000;
        text-shadow: 1px 1px 20px #000000;
        border: solid #337FED 0;
        text-decoration: none;
        display: inline-block;
        cursor: pointer;
        text-align: center;
    }

    .button-style:hover {
        border: solid #337FED 1px;
        background: #1E62D0;
        background-image: -webkit-linear-gradient(top, #1E62D0, #3D94F6);
        background-image: -moz-linear-gradient(top, #1E62D0, #3D94F6);
        background-image: -ms-linear-gradient(top, #1E62D0, #3D94F6);
        background-image: -o-linear-gradient(top, #1E62D0, #3D94F6);
        background-image: -webkit-gradient(to bottom, #1E62D0, #3D94F6);
        -webkit-border-radius: 6px;
        -moz-border-radius: 6px;
        border-radius: 6px;
        text-decoration: none;
        border: solid #337FED 0;
    }
    </style>
    </head>
    <body>
        <h1>Hello World dApp</h1>
        <br/>

            <div id="output_area">
                    <h2>Current Message</h2>
                    <dd>
                            <h3 id="data_output"></h3>
                    </dd>
            </div>

        <div style="border:1px solid black;width:300px;">
                    <h4 style="position:relative;top:-20px;left:3px;"> Change Message </h4>
                    <div style="margin-left:20px;margin-right:20px;margin-bottom:20px;">
                            <form id="form" style="margin-top:-20px;">
                                    <input id="input" type="text" style="width:250px;"/> <br>
                                    <br>
                                    <input class="button-style" type="submit" value="New Message" />
                            </form>
                    </div>
            </div>

        <script type="text/javascript" src="node_modules/jquery/dist/jquery.min.js"></script>
            <script>
                    $("#output_area").hide();
            </script>
        <script type="text/javascript" src="node_modules/web3/dist/web3.min.js"></script>
        <script type="text/javascript" src="src/app.js"></script>
    </body>
    </html>
```

and

The file that all the work happens in:

```
 1: // getWeb3 connects to MetaMask with the local server and starts the web3 library.
 2: // Function from MetaMask Documenation.
 3: const getWeb3 = () => {
 4:     return new Promise((resolve, reject) => {
 5:         window.addEventListener("load", async () => {
 6:             if (window.ethereum) {
 7:                 const web3 = new Web3(window.ethereum);
 8:                 try {
 9:                     // Get the users accounts - request permission to use them in MetaMask
10:                     await window.ethereum.request({ method: "eth_requestAccounts" });
11:                     resolve(web3);
12:                 } catch (error) {
13:                     reject(error);
14:                 }
15:             } else {
16:                 reject("Please Install MetaMask - Required for this web-application to work.");
17:             }
18:         });
19:     });
20: };
21:
22: // Get access to a contract - pulls in the ABI.
23: // Function from MetaMask Documenation.
24: const getContract = async (web3) => {
25:     const ABI = await $.getJSON("./contracts/SavedMessage.json");
26:
27:     const netId = await web3.eth.net.getId();
28:     const deployedNetwork = ABI.networks[netId];
29:     const contrctBindABI = new web3.eth.Contract( ABI.abi, deployedNetwork && deployedNetwork.address );
30:     return contrctBindABI;
31: };
32:
33: // displayData will show modified data on the user interface.
34: const displayData = async (contractHandle, contract) => {
35:     contractHandle = await contract.methods.getCurrentMessageTxt().call();
36:     $("#output_area").show();
37:     $("#data_output").html(contractHandle);
38: };
39:
40: // bindToForm assicates functions with a form on the web page.
41: const bindToFormX = (contractHandle, contract, accounts) => {
42:     let input;
43:
44:     // Collect new input.
45:     $("#input").on("change", (evnt) => {
46:         input = evnt.target.value;
47:     });
48:
49:     // Handle form submission.  Call the contract to change the message.
50:     $("#form").on("submit", async (evnt) => {
51:         evnt.preventDefault(); // Stop form from taking "action" and submitting to server.
52:         await contract.methods.setCurrentMessageTxt(input).send({ from: accounts[0], gas: 40000 });
53:         // updateGreeting is the method in the contract
54:         displayData(contractHandle, contract); // Display modified data.
55:     });
56: };
57:
```

```
58: // runApp will:
59: //    1. get connection to accounts (getWeb3)
60: //  2. get set of accounts.
61: //  3. display current data.
62: //  4. Setup to handle form.
63: async function runApp() {
64:     let contractHandle;
65:
66:     const web3 = await getWeb3();
67:     const accounts = await web3.eth.getAccounts();
68:     const contract = await getContract(web3);
69:
70:     displayData(contractHandle, contract);
71:     bindToFormX(contractHandle, contract, accounts);
72:
73:     setInterval(function(){
74:         displayData(contractHandle, contract);
75:     }, 1000);
76: }
77:
78: runApp();
```

Running a “server”

```
$ npm run start
```

or with a Go server

```
$ cd simple-go-server
$ go build
$ cd ../client
$ ./simple-server/simple-go-server --dir . --port 3004
```

Interacting with the dApp in a browser

Demo...