# Pre-requisites

## Install Python, Nodes.js

For now, install python 2.7 and pip. Include python and pip to you environment PATH

(e.g. ;C:\dev\Python27;C:\dev\Python27\Scripts)

## Install Nodes.js and NPM

npm install eth-ens-namehash -g

npm install web3 -g or npm install web3 -S

npm install webpack -g

## Unit Tests using Truffle and Testrpc

IMPORTANT: You have to different your Codes (client-side or server-side) in production and test environment. This is because Transactions must be mined in Production and Testnet. Transactions like creating new contracts should be performed in a separate workflow requiring account password.

Using Testrpc or Truffle, you do not need to wait for your transactions to be mined.

testrpc -p 8700

Run “truffle” in Windows Powershell

Run “truffle migrate” to build and deploy your smart contracts

Run “truffle test” to execute your unit tests

# DApp – Web UI

## Project Root directory

<DIR> ..

<DIR> build

<DIR> contracts

<DIR> migrations

<DIR> node\_modules

<DIR> src

<DIR> test

<DIR> www

package-lock.json

package.json

truffle-config.js

truffle.js

webpack.config.js

## Install and Run your DApp

From project Root directory:

npm install

npm run dev

## Using Testrpc

Get the list of available accounts from Testrpc:

Available Accounts

==================

(0) 0x69bc681fa9b90dd59147044bb831cf0e3649440b

(1) 0xab7fbd5a3d6366649caf5fef5ed3abacd5907f04

(2) 0x4954867119ca2477f313d53319ce091bee615c76

(3) 0x105d68a8db49759f8a5fc22e4ac26b77d24437dc

(4) 0x5082c43f7b5167e63e9ce5cb71328282bbfd3586

(5) 0x2ddd2bac7cb30b3cba12ba06a976614597026fbf

(6) 0x51fbca896ff41e7d2e92da863b35b2f6f0eb4896

(7) 0x1e59657ad8d81b9e0fb2e575b0de9aa1c278af04

(8) 0xdebb60de22f6f2ceaba9addca27ba19d85d8abcc

(9) 0x788a06a29db4106a858b4a9e2d5484051787b19a

Note that there is no need to personal.unlockAccount(…) when using testrpc as your Ethereum client. See src/dapp.js

window.addEventListener("load", function() {

if (typeof web3 !== "undefined") {

window.web3 = new Web3(web3.currentProvider);

} else {

// using geth 8545

//window.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

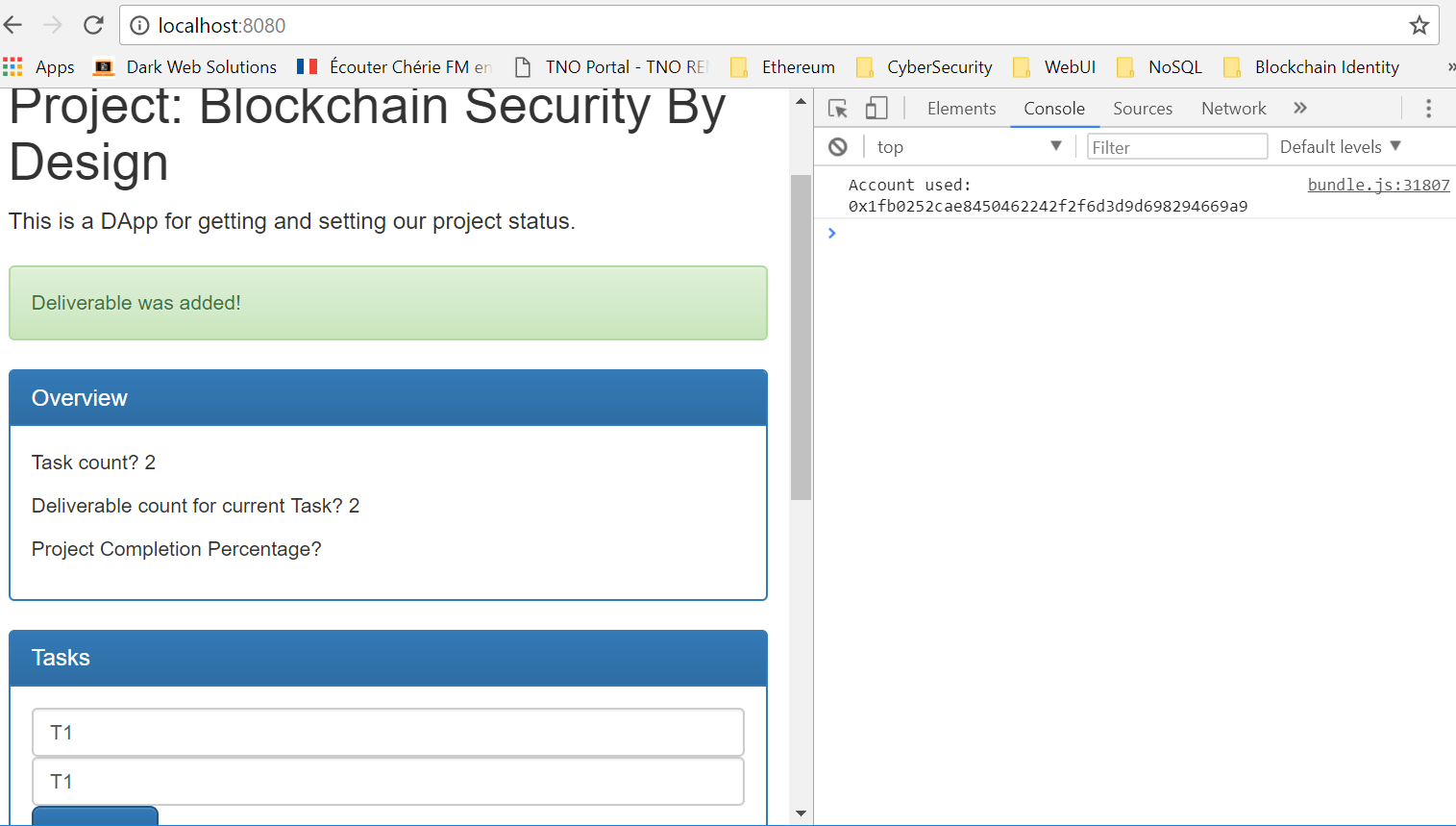
// using testrpc -p 8700

window.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8700"));

}

## Running DApp

From web brower, open <http://localhost:8080/>



## dapp.js

Refer to src/dapps for the Web3 APIs and/or Smart Contract functions used.

# DApp - Event Monitoring

## Project directory

DIR> ..

<DIR> src

<DIR> www

package-lock.json

package.json

truffle.js

webpack.config.js

Open truffle.js, this project references the same contracts from above project’s build directory.

It also connects to the same Ethereum client port to access the contract(s).

module.exports = {

contracts\_build\_directory: "..\tnoproject\build",

networks: {

development: {

host: "localhost",

port: 8700, // run testrpc -p 8700, default was 8545

Check src/dapp.js

var TNOProject = contract(require("../../tnoproject/build/contracts/TNOProject.json"));

## Deploy to specific Host and Port

Check package.json,

"scripts": {

"build": "webpack",

"dev": "webpack-dev-server --host localhost --port 8081"

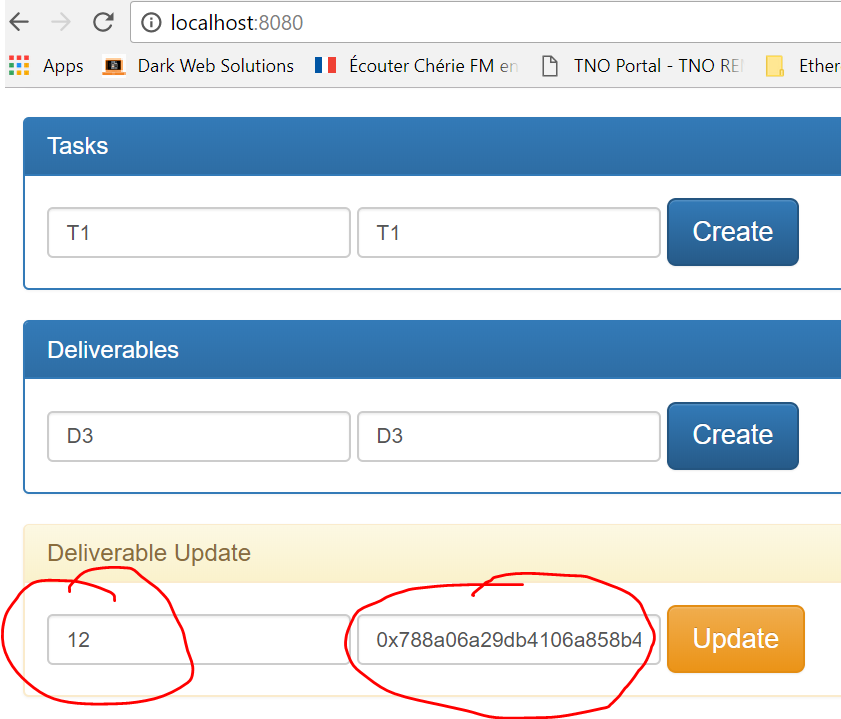
},

Note that this DApp will use port 8081

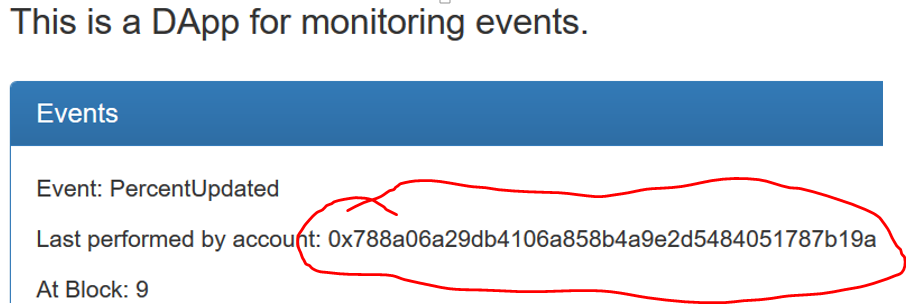
## Running DApp

Updates from DApp <http://localhost:8080/> will trigger contract Events.

Shown below, enter percentage update and the Account used for this update. (Please refer to the list of Available Accounts in Testrpc)



Open from web browser <http://localhost:8081/>



## dapp.js

Refer to src/dapps for the Web3 APIs and/or Smart Contract functions used.

# Attaching consoles to Testrpc

## Truffle Console

You can connect to Testrpc by running “**truffle console**” command from Windows Powershell

PS C:\dev\ethereum\tno\tnoproject> truffle console

truffle(development)>

## Geth

C:\dev\ethereum\tno\tnoproject>**geth attach rpc:http://localhost:8700**

Welcome to the Geth JavaScript console!

instance: EthereumJS TestRPC/v1.1.3/ethereum-js

coinbase: 0x9c1c6c57683315b87007d281d2144582736953d1

at block: 9 (Thu, 28 Dec 2017 15:22:07 +08)

modules: eth:1.0 evm:1.0 net:1.0 personal:1.0 rpc:1.0 web3:1.0

> eth.accounts

["0x9c1c6c57683315b87007d281d2144582736953d1", "0xa69f050534a1ff62364af06cc46d55b1f5afe332", "0x87ca2de31cd3299c7b46aa79e17754e9386dba57", "0xdc478759c7ea5ffff7e33d713333742302579f60", "0xb23d68b9faf400b693b6e1144c952e9a162515b0", "0x0932947af62ce735e9460d15c91befc95de0a95d", "0xe83aa46f30e0c67cf1ad15528df17bda9ee169c6", "0x64ddc3a5b79001ee0beb3fd906533b6c3fd3ffeb", "0x912e018c375becbf76ea8871943365ccf615d692", "0x32636d0eab0ac2f587968d07194865f848361a3c"]

> eth.blockNumber

9

>

### Loading JS from Geth console

> loadScript("/src/helloWeb3.js")

hallo Web3

99032922700000000000

99032922700000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

100000000000000000000

true

>

## Node.js

Required modules must exist in ./node\_modules directory.

If not, run “npm install” in root directory where package.json can be found.

Finally, run “node”.

From node console, enter the Javascripts statements from helloWeb3.js, e.g.:

var Web3 = require("web3");

console.log("hallo Web3");

if (typeof web3 !== 'undefined') {

web3 = new Web3(web3.currentProvider);

} else {

// set the provider you want from Web3.providers

web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8700"));

}

var coinbase = web3.eth.coinbase;

var balance = web3.eth.getBalance(coinbase);

console.log(balance);

web3.eth.accounts.forEach(function(account) {

balance = web3.eth.getBalance(account);

console.log(balance);

});

# DApp – Ether Transfer

## Project Directory

<DIR> ..

<DIR> src

<DIR> www

package-lock.json

package.json

readme.txt

Run “npm install” (this will install dependencies from package.json)

Or

Run install individually, e.g:

npm install -g watchify

see https://github.com/browserify/watchify

npm install --save-dev babelify

...

## package.json

Note in package.json

..."main" is configured as:

{

"name": "ETH Sender App",

"version": "1.0.0",

"description": "ETH Sender App - interacting-with-the-blockchain",

"**main**": "**../src/dapp.js**",

...

..."start" is configured as:

"scripts": {

"**start**": "**watchify src/dapp.js -o www/bundle.js -v**"

},

## Running DApp

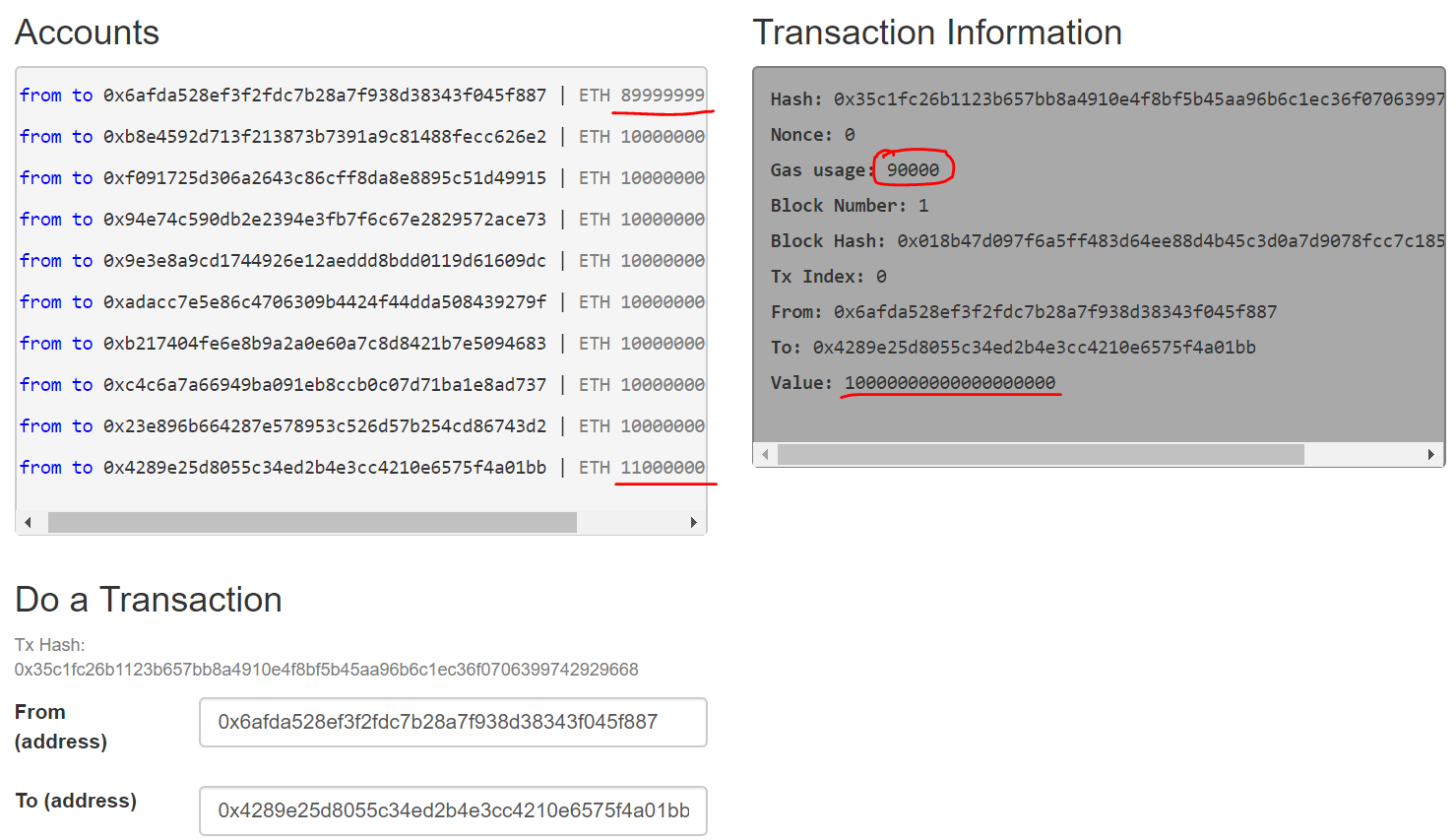
Run “testrpc -p 8700”

From project root directory, run “npm start”

From web browser, open www/index.html

Just click on the Account address of from link and to link to transfer Ethers.

Note that the transaction’s Gas usage is also deducted from the sender’s account. (This what the mining node would earn.)



## dapp.js

Refer to src/dapps for the Web3 APIs and/or Smart Contract functions used.

# DApp – Token App

## Project Directory

<DIR> ..

<DIR> bin

<DIR> contracts

<DIR> src

<DIR> www

package-lock.json

package.json

readme.txt

## package.json

Note again the directories in package.json

{

…

**"main": "../src/dapp.js"**,

…

"scripts": {

"start": "watchify src/dapp.js -o www/bundle.js -v"

},

## Running DApp

Run “testrpc -p 8700”

From project root directory, run “npm start”

From web browser, open www/index.html

## Deploying Smart Contract instance from file

If you have not installed dependencies, run “npm install” in the directory where package.json is located.

Run “node”.

From node console, execute the statements from src/deploy.js

The statements

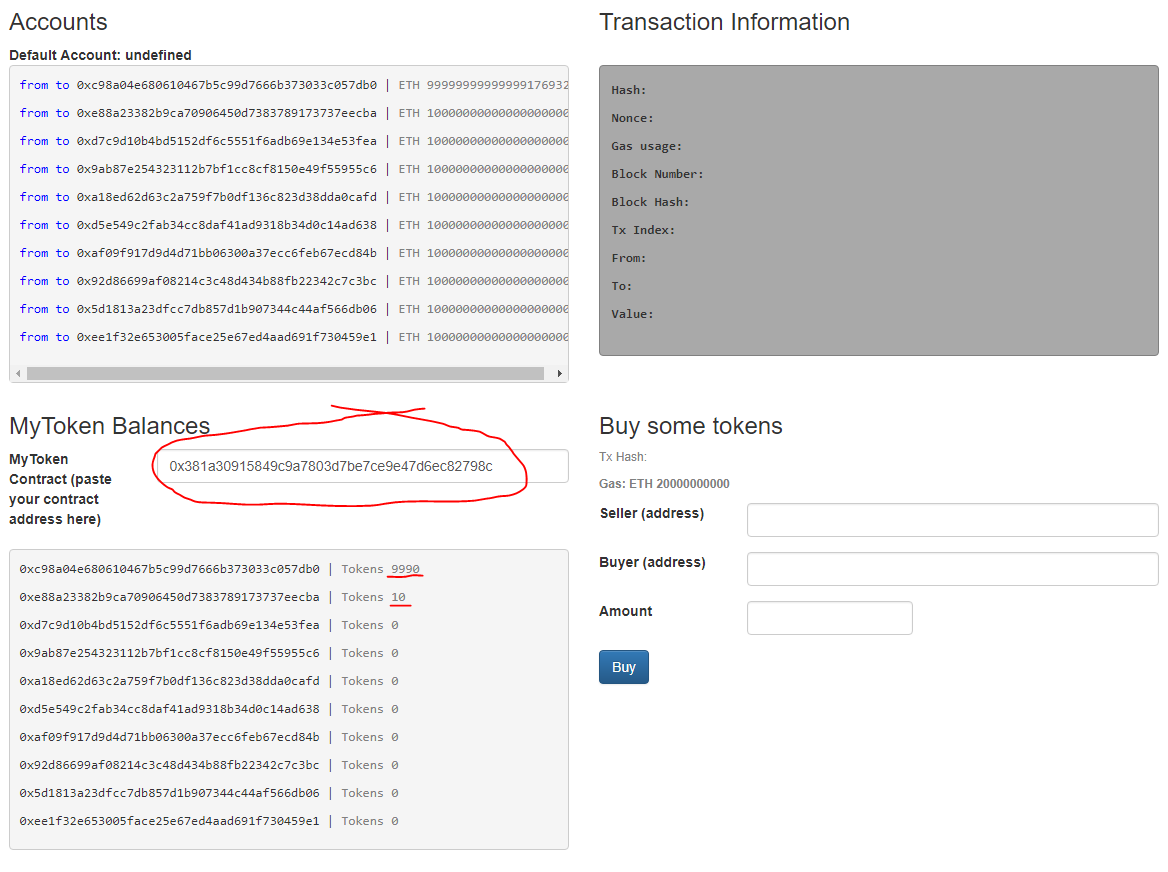
* deploys an instance of SebToken smart contract (with initial balance for accounts[0]) to the Testrpc
* transfers 10 tokens from account[0] to account[1]

Copy the contract address

> contractAddress = deployedContract.address;

'0x381a30915849c9a7803d7be7ce9e47d6ec82798c'

From Web UI www/index.html, paste the contract address to the My Token contract address field

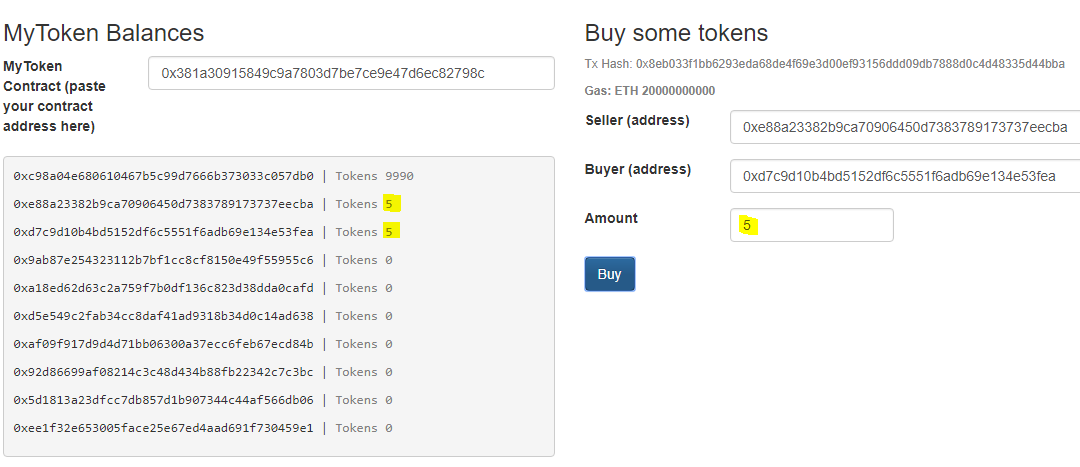


## Transferring Your Application Tokens

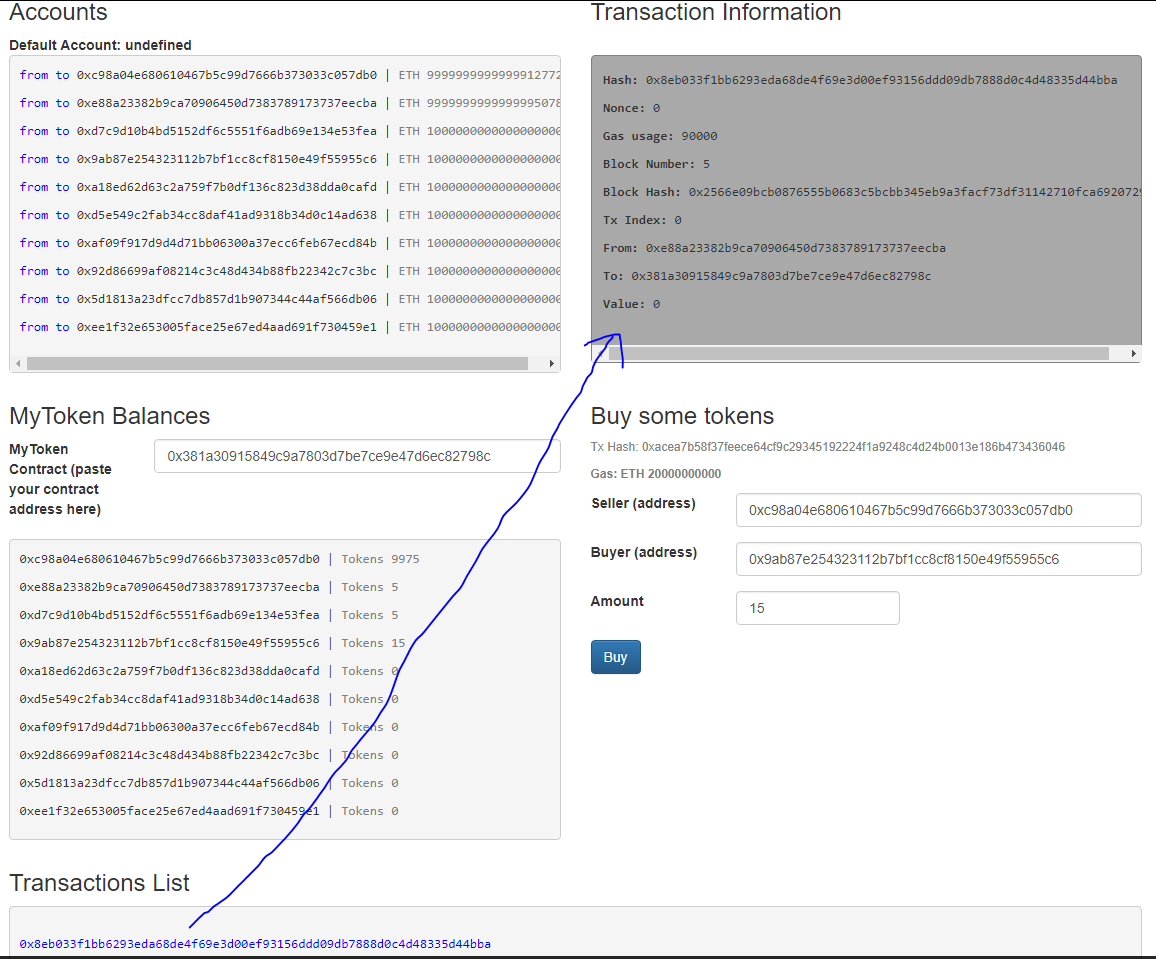
From the same Web UI with www/index.html

Just click on the Account address of from link and to link to transfer tokens, the click on Buy button.

The token amount will be transferred. (Sender must have sufficient balance to do the transfer)



Clicking on the Transaction Hash will refresh the Transaction Information (see below screen shots)



Please refer to the Web3 APIs used in **src/deploy.js** and **src/dapp.js**

# Known Issues

Problem

<contract> has not been deployed to detected network

Solution

Run truffle migrate again

Problem

truffle migrate Exceeds block gas limit

Solution

https://stackoverflow.com/questions/42924634/unable-to-run-truffle2-1-2-migrate-network-live-exceeds-block-gas-limit

Add to truffle.js

gas: 3000000, // use web3.eth.getBlock("pending").gasLimit >>4712388

Problem

<contract> has not been deployed to detected network

Solution

Run truffle migrate again

# References

<https://github.com/dickolsson/irondoers>

<https://dickolsson.com/building-dapps-on-ethereum-part-3-user-interface/>

<https://github.com/angular/angular-cli/issues/3765>

<https://stackoverflow.com/questions/33272967/how-to-make-the-webpack-dev-server-run-on-port-80-and-on-0-0-0-0-to-make-it-publ>

<https://blog.zeppelin.solutions/a-gentle-introduction-to-ethereum-programming-part-1-783cc7796094>

<https://blog.zeppelin.solutions/a-gentle-introduction-to-ethereum-programming-part-2-7bbf15e1a953>