System and Unit Test Report

Ethereum Energy Exchange

The 2x2

December 3, 2017

## Scenario

1. Sprint 1
   1. User Story 1: As a developer, I want to learn about block-chaining with Ethereum and cryptocurrency.
   2. User Story 2: As a developer, I want to learn what framework would work best with smart contract implementation and its relation to SEADS’ purpose.
   3. Scenario:
      1. No level of action needed for this particular sprint as we were trying to understand and figure what cryptocurrency was and its importance in SEADS.
2. Sprint 2
   1. User Story 1: As a developer, I want to be able to buy and sell energy with Ethereum.
   2. User Story 2: As a developer, I want to fundamentally understand how smart contract and transaction will interact in a practical application

* Scenario
* We were able to test the sending of cryptocurrency through TestBank.sol, a given test currency within truffle.
* Steps to run test.
  + -> testrpc (in a seperate terminal console)
  + -> truffle console
  + ->truffle migrate
  + -> truffle test /path/to/test/file
  + Expected output -> “Should transfer one account to another” (green text)
* NOTE - This test has been overwritten, but if needed old test screenshots are available with documentation in weekly report.
* This scenario also works for 2b because it shows how smart contracts are used in a “practical application”.

1. Sprint 3
   1. User Story 1: As a developer, I want to develop a DAPP to host our smart contract and deploy our contracts
   2. User Story 2: As a user, I want to be able to interact with the smart contract and perform transactions
   3. User Story 3: As a developer, I want to combine our smart contract with our DAPP so that we can test the contract on a simulated testing environment.
   4. Scenario
      1. We did were not able to develop a DAPP as it required a greater understanding of React when none of us knew it. We ran into some issues when trying to interact with the smart contract in Parity so the scenario testing will be incomplete here until Sprint 4.
      2. Start parity, type “parity ui --chain=ropsten”
      3. Click “contract” tab and follow by “develop” and paste contract into it
      4. Click compiler version “0.4.18” and “deploy”
      5. User should see at the bottom right that the transaction contract is being sent to the blockchain and being mined for confirmation
      6. Under “contract” tab, the contract should appear and select a function and address to interact with.
      7. Press “execute” which looks like an arrow in order to select a function that requires gas to run on the blockchain.
      8. User should see their transactions being confirmed and mined and the result should appear under the “event” log
         1. However, we could not get this far as we were running into issues on understanding the underlying aspect of writing a smart contract.
2. Sprint 4
   1. User Story 1: As a developer, I want to amend the existing project to be able to monitor and interact with token currency through the blockchain.
   2. User Story 2: As a user, I want a simple-to-use interface to monitor and manage my energy portfolio.
   3. User Story 3: As a developer, I want to provide accurate timestamps for energy microtransactions.
   4. User Story 4: As a user, I want to be able to set a time period to sell back energy.
   5. Scenario
      1. The idea here is the same as Sprint 3 but with more improvements on vii.
         1. Starting at vii, user should be able to see the result of the transaction for each function under the event log.
         2. In order to see previous results without the event log, user could use the getter functions above the “event” log section and instantly see the results.
      2. Following steps will setup the EthereumEnergyExchange Dapp on Parity with Dapp code pulled from our repo.
         1. Clone our git repo EthereumEnergyMicrotransactions.  
            <https://github.com/zdolson/EthereumEnergyMicrotransactions.git>
         2. Once cloned, all necessary files should be in place for the dapp, now to make the dapp visible to Parity enter one of the following commands depending on your OS.
            1. Mac: ln -s $PWD/dist $HOME/Library/Application\ Support/io.parity.ethereum/dapps/mydapp
            2. Linux:
            3. Windows:
         3. Once the link has been created in the correct directory for Parity to see, start Parity on the Ropsten testnet with the following terminal command:  
            parity ui --chain ropsten
         4. Upon launching the previous command, a website will open up in your default browser. On this website, proceed to the Applications section. Then you should see the