System and Unit Test Report
Ethereum Energy Exchange
The 2x2
December 3, 2017

Scenario

1. Sprint 1
    a. User Story 1: As a developer, I want to learn about block-chaining with Ethereum and cryptocurrency.
    b. 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.
    c. Scenario:
        i. 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
    a. User Story 1: As a developer, I want to be able to buy and sell energy with Ethereum.
    b. 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".
        -
3. Sprint 3
    a. User Story 1: As a developer, I want to develop a DAPP to host our smart contract and deploy our contracts
    b. User Story 2: As a user, I want to be able to interact with the smart contract and perform transactions

c. 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.

d. Scenario

    i. 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.

    ii. Start parity, type "parity ui --chain=ropsten"

    iii. Create an account under "account" tab and follow the instruction and keep tabs on the mnemonic words.

    iv. Click "contract" tab and follow by "develop" and paste contract into it

    v. Click compiler version "0.4.18" and "deploy"

    vi. User should see at the bottom right that the transaction contract is being sent to the blockchain and being mined for confirmation

    vii. Under "contract" tab, the contract should appear and select a function and address to interact with.

    viii. Press "execute" which looks like an arrow in order to select a function that requires gas to run on the blockchain.

    ix. 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.

4. Sprint 4

    a. 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.

    b. User Story 2: As a user, I want a simple-to-use interface to monitor and manage my energy portfolio.

    c. User Story 3: As a developer, I want to provide accurate timestamps for energy microtransactions.

    d. User Story 4: As a user, I want to be able to set a time period to sell back energy.

    e. Scenario

        i. 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.

        ii. **Disclaimer: If you are attempting to use the functions inside of the dapp, you must have a sufficient amount of Ether in your wallet.**

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.
   a. Mac: `ln -s $PWD/dist $HOME/Library/Application\ Support/io.parity.ethereum/dapps/mydapp`
   b. Linux: `ln -s $PWD/dist $HOME/.local/share/io.parity.ethereum/dapps/mydapp`
   c. Windows: `mklink /D "%APPDATA%/Parity/Ethereum/dapps/mydapp" "%cd%/dist"`
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.
   a. On this website, proceed to the Applications section.
   b. Click on the SEADS ETHEREUM ENERGY EXCHANGE Dapp.
5. Features available on the dapp:
   a. Energy Unit (Enit) faucet for adding energy to your account.
      i. This will add energy to your default account which you create on first launch of Parity.
   b. Sell Enits to another registered person.
      i. If the person is not registered, the transaction will throw and no Enits will be transferred; however, the transaction fee (gas) will still be deducted from the default accounts Ether balance.

    ii. In order to execute the transaction, enter in the number of Enits (1 Enit = 1 kWH) you would like to sell. Then provide the username of the buyer of your energy. Once filled in, click the Sell Energy Button below.
    iii. Once the button has been clicked, Parity will pop up requesting your approval of the transaction. This is due to the use of your Ether to pay for the transaction gas.

c. Register an account.
    i. To register an account with the contract's registry, simple provide a new username and a wallet's hex address.
    ii. Click the Register button and provided the input you provided is correct, the username will be registered with the provided address.