Netvrk – Solidity Task Documentation

Task:

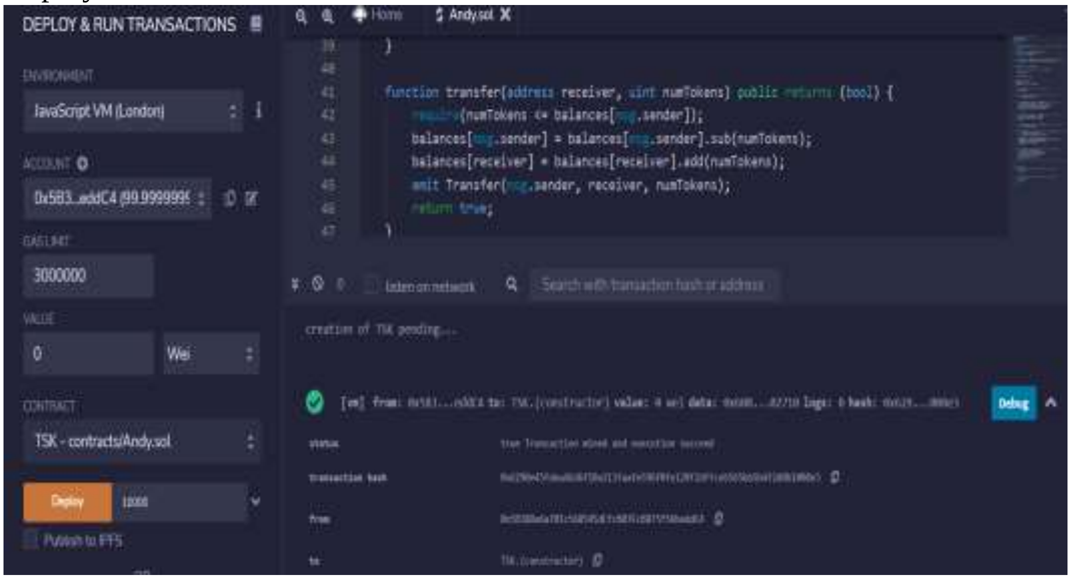
1. Create a new token.
2. Write a smart contract to swap Ether for the token created in the previous step.
3. Create a DApp for the above use case.

Assumptions Made:

1. The new token that I have created is an ERC-20 token.
2. The name of the token is "Task – Decentralized". The symbol is "TSK" and I have taken two decimal places.
3. I have assumed the cost of the newly created token to be 1 ETH.

Solution:

1. Smart Contract
   - I have used the basic and standard functions and events to create the ERC-20 token.
   - Upon creating the token, I have assigned the total supply to address of the smart contract.
   - I have one function named swapFromETH() that takes one parameter as an input and then swaps ETH to the newly created tokens.
   - In line 72 of the code,
     `tokens = (amount/TokenCostInETH)*(10**decimals);`
     `amount` is the ETH that is to be swapped (passed in as an input parameter), `TokenCostInEth` is the cost of the token in ETH. I have multiplied this with 10^(the number of decimal places of the newly created token).
   - Deploy:

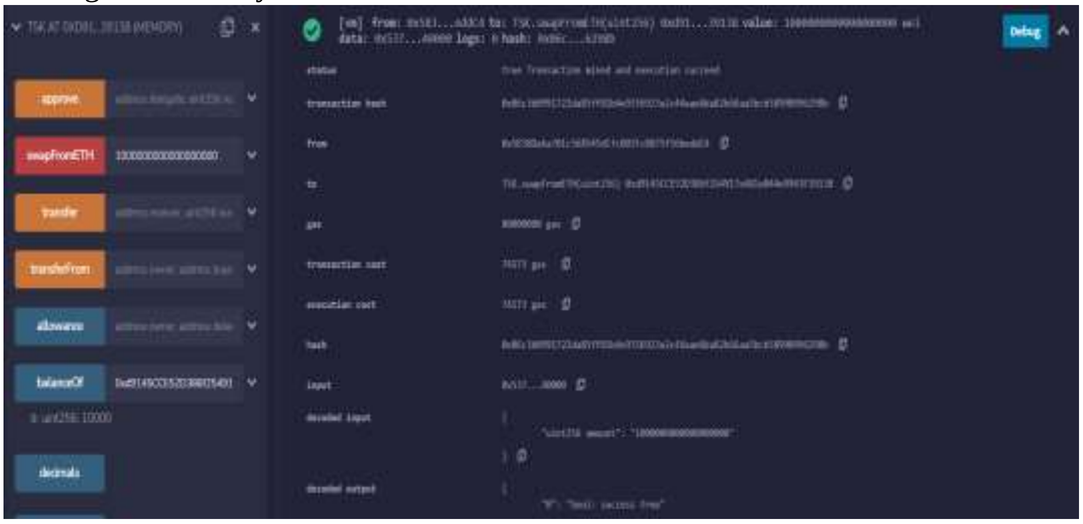


```
    }

    function transfer(address receiver, uint numTokens) public returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
```

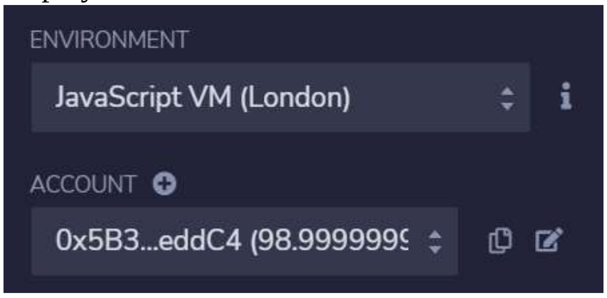
```
creation of TSK pending...

[vm] from: 0x5B3...eddC4 to: TSK.(constructor) value: 0 wei data: 0x608...[illegible] logs: 0 hash: 0x[illegible]
status              true Transaction mined and execution succeed
transaction hash    [illegible]
from                [illegible]
to                  TSK.(constructor)
```

Successfully deployed the contract and the address of the contract is: 0xd9145CCE52D386f254917e481eB44e9943F39138.

– Testing functionality:

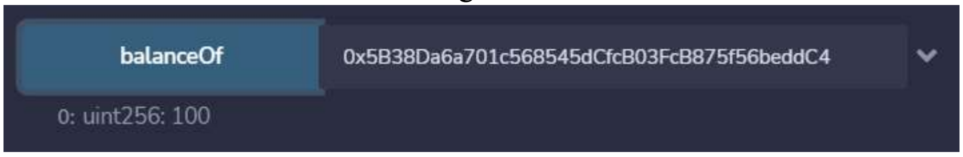


Execution of the swapFromETH() function. The above image shows the input as "1000000000000000000" (i.e. 1 ETH) and the output as true.
In the previous image, the balance of the msg.sender can be seen as 99.9 ETH, but after the execution of this function, the balance is updated to 98.99 ETH as displayed below.



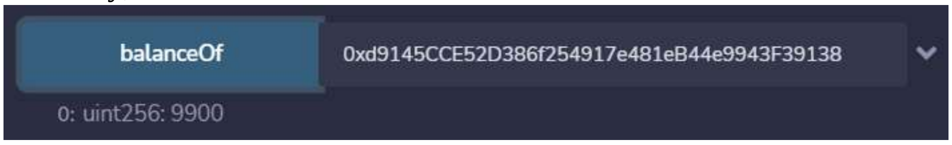
ENVIRONMENT
JavaScript VM (London)
ACCOUNT
0x5B3...eddC4 (98.999999

The balance of the tokens in the msg.sender's wallet is as follows:



balanceOf 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
0: uint256: 100

The "100" is nothing but 1 TSK (due to the 2 decimal places that I have assumed).
Similarly, the balance of the smart contract is as follows:

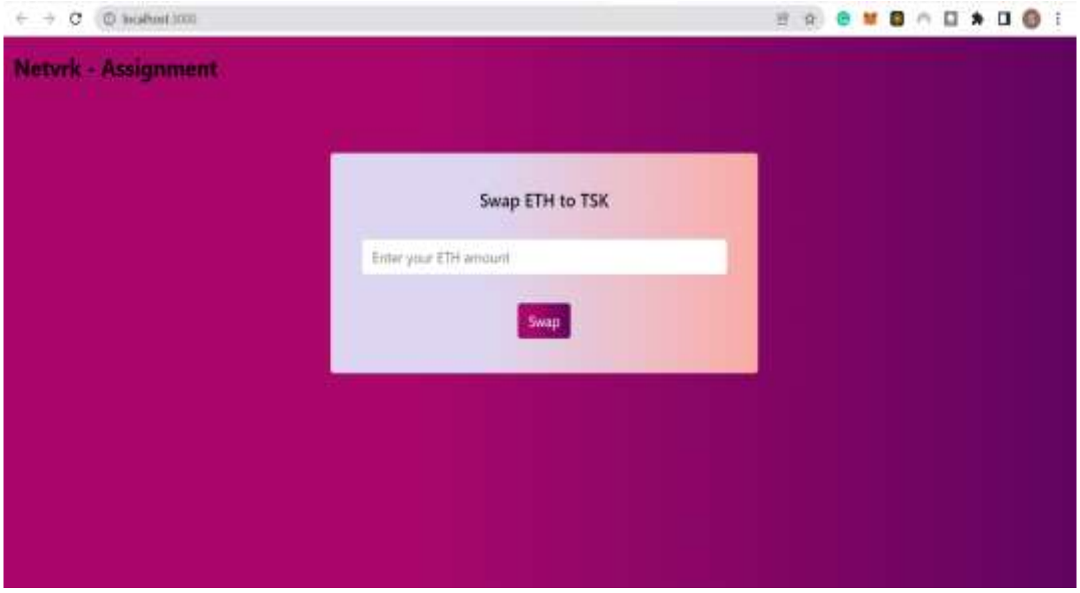


balanceOf 0xd9145CCE52D386f254917e481eB44e9943F39138
0: uint256: 9900

The "9900" is nothing but 99 TSK.

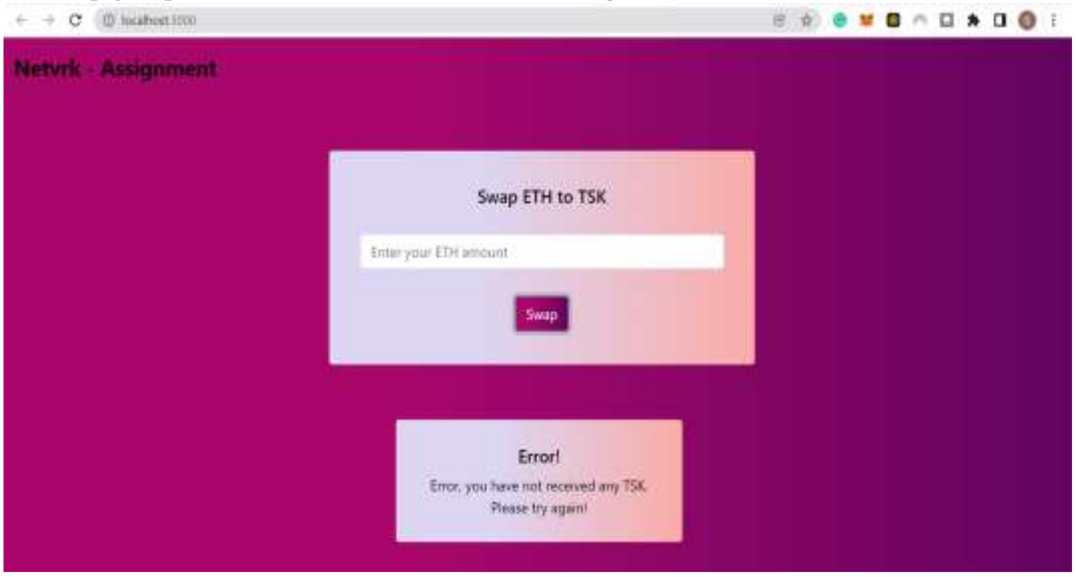
2. Unit Testing
   I have done the testing manually as I don't know how to do the unit testing. I'm yet to learn it.
3. DApp
   – I have used React for the creation of the DApp.
   – I began by using the following syntax: "`create-react-app netvrk-dapp`".
   – I installed the required npm modules.

- I then used the react components like the card and created the UI as per the requirement, displayed below:



- Then I added the smart contract to the current project directory along with the compile.js and deploy.js.
- The code in the compile.js was for the older versions of solc and so I made the required modifications to it and executed both the files.
- After executing the deploy.js file, the smart contract was deployed but for some reason the TSK tokens created were not being assigned to the address of the smart contract. So, I went to the remix IDE and deployed the smart contract again via Metamask.
- Now the smart contract was deployed properly and the total supply was assigned to the address of the smart contract: 0xcd4318f7d5bc40037b854f99de13369ef993b543.
- Then I started working with Web3 to add a bridge between the deployed smart contract and the frontend.
- I faced another issue here as webpack was updated recently and there was a break while trying to execute the web3 functions. To solve this, I took help from the following link and found a workaround to it.
- Now I was able to access and execute the web3 functions without any error.
- Once I was able to interact with the smart contract from the frontend and execute functions successfully, I also added some code for error handling.

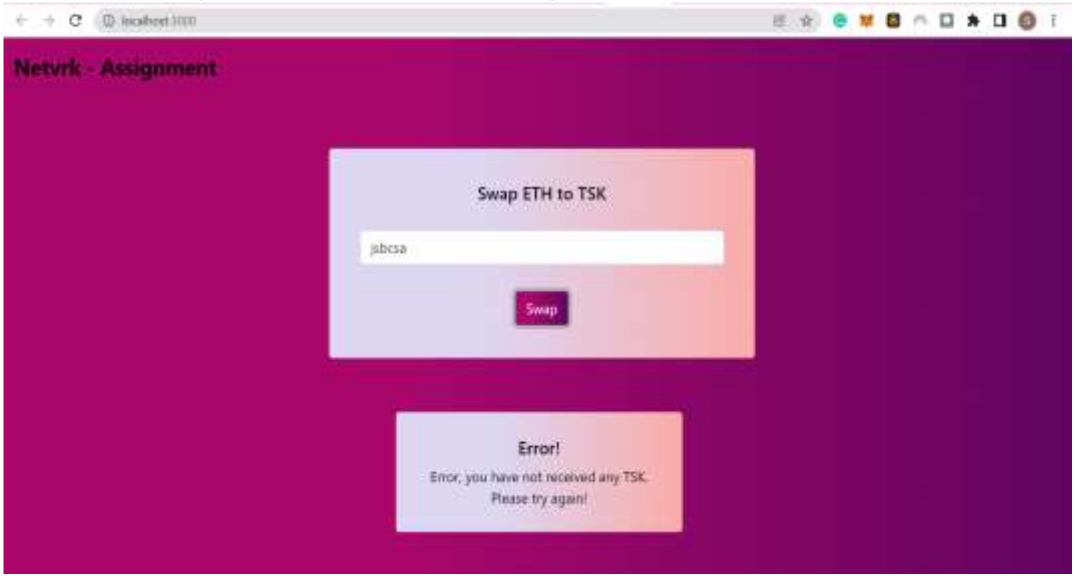
- An empty input field would result the following:



Error!

Error, you have not received any TSK. Please try again!

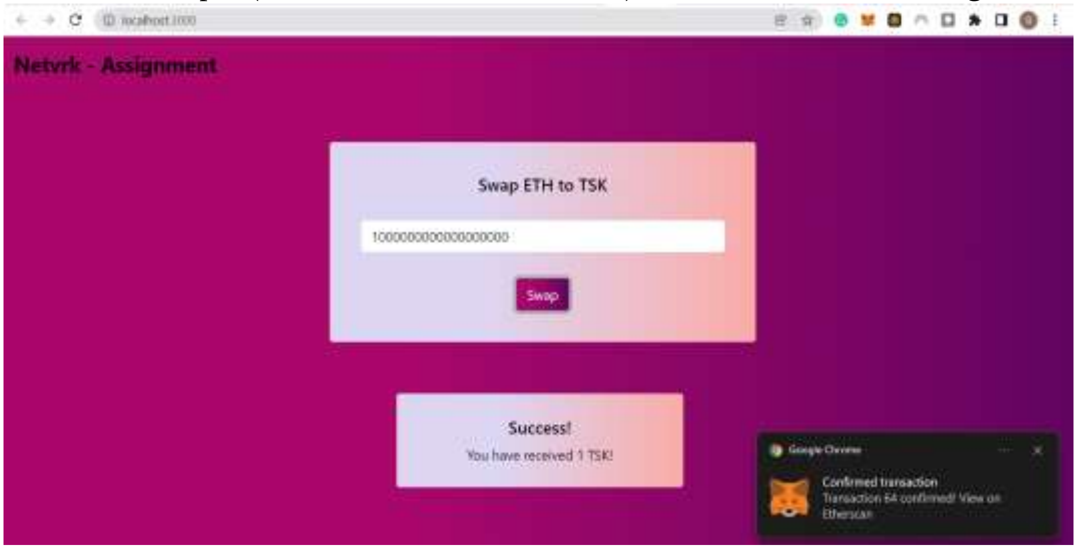
- An invalid entry would result the following:



Error!

Error, you have not received any TSK. Please try again!

- A successful input (1000000000000000000 Wei) would result the following:



Success!

You have received 1 TSK!

Confirmed transaction

Transaction 64 confirmed! View on Etherscan