**CSED490U Blockchain & Cryptocurrency**

**Assignment 7**

****

**Submitted by- Sajan Maharjan**

**POVIS id-** [**thesajan@postech.ac.kr**](mailto:thesajan@postech.ac.kr)

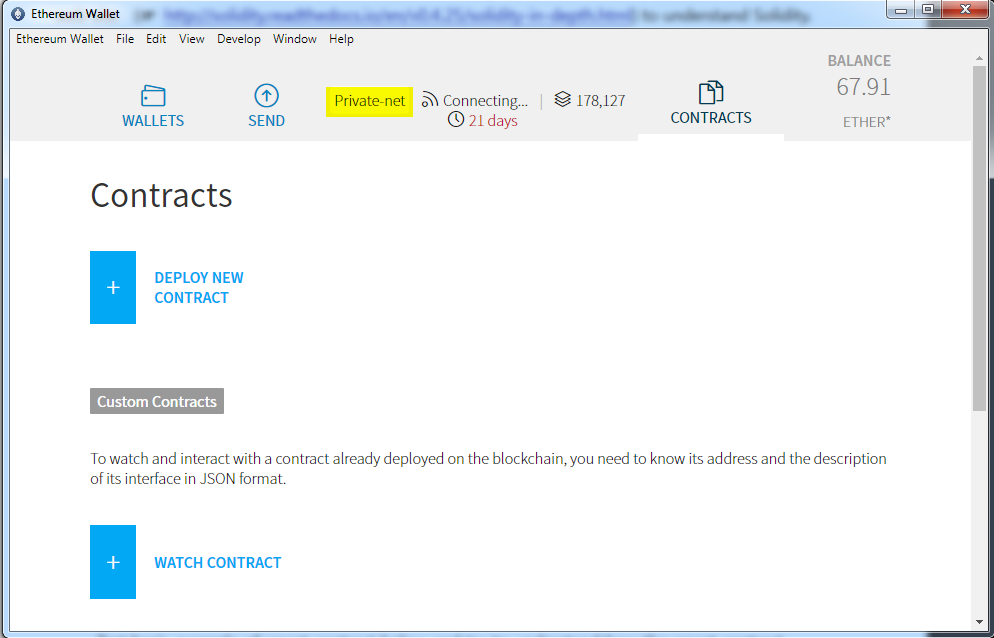
**Registration Number- 20182095**

**<Step. 1> Install Ethereum wallet / Mist**

> Ethereum wallet was installed for the Windows operating system using the link- <https://github.com/ethereum/mist/releases>

**<Step. 2> Connect to your own private network**

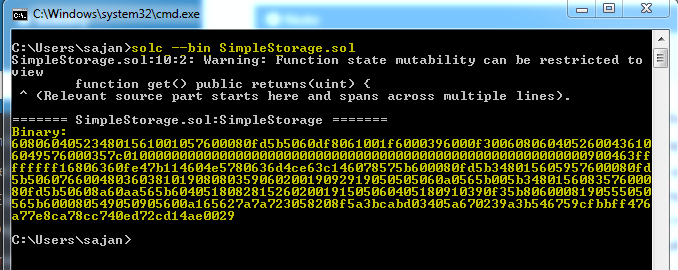
> Geth was initially launched and connected to a private network. Next, ethereum wallet application was launched and ethereum wallet was instantly connected to the operating instance of geth. i.e.



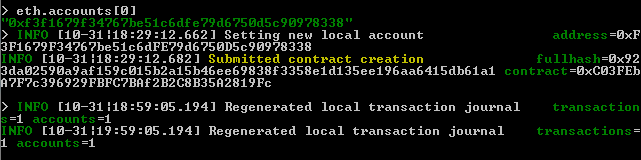
**<Step. 3> Create a simple smart contract**

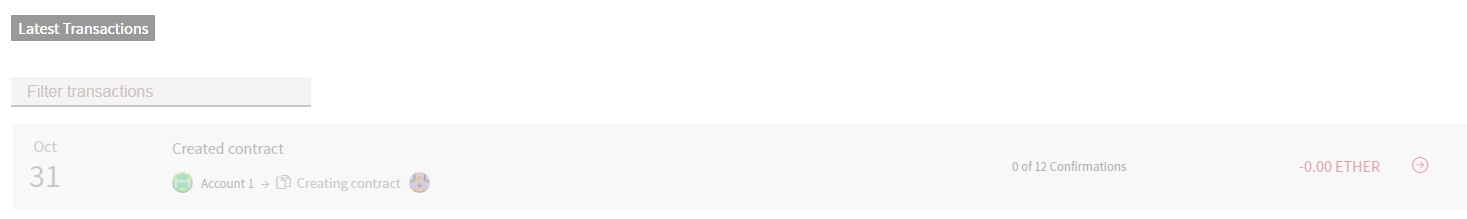
**3-1 Create new smart contract that contains the code below and compile it**

The given source code was saved into a solidity contract SimpleStorage.sol and compiled successfully to generate the contract byte code as shown below-



**3-2) If you compile successfully, publish the smart contract on the private network**



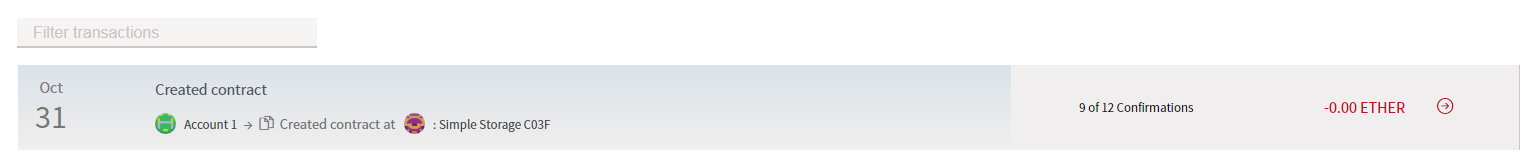


From the ethereum wallet snapshot shown above, it can be confirmed that the smart contract was sent to the ethereum blockchain but was not yet confirmed, as mining was not yet initiated.

**3-3) After the block contains the transaction generated by the corresponding smart contract is mined, check the status of the smart contract in detail. Capture the status and attach it to the report.**

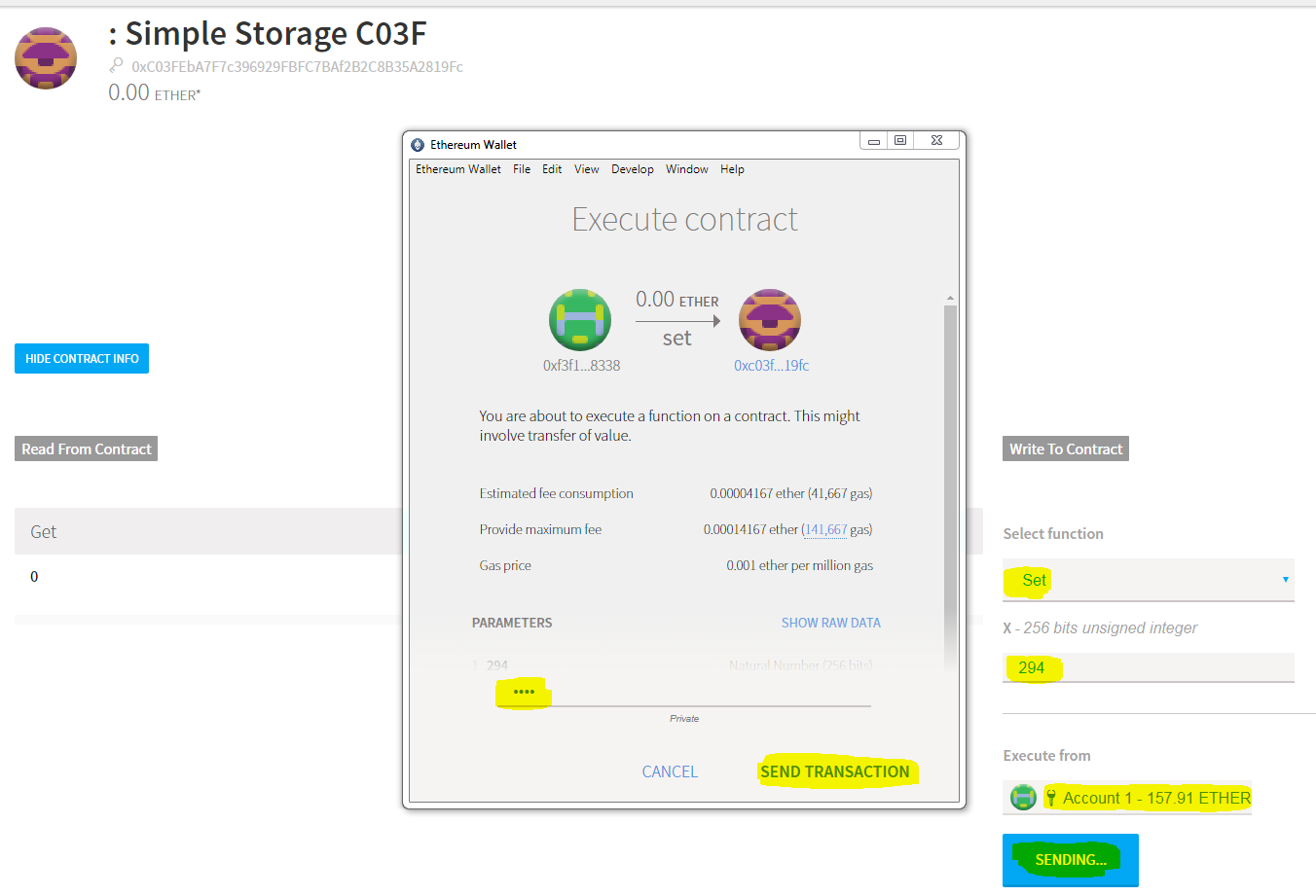
> miner.start() command was used to start mining and confirmation on the contract were increased slowly

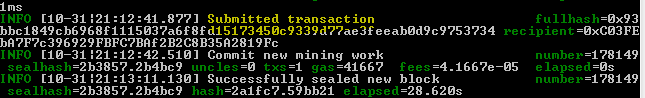




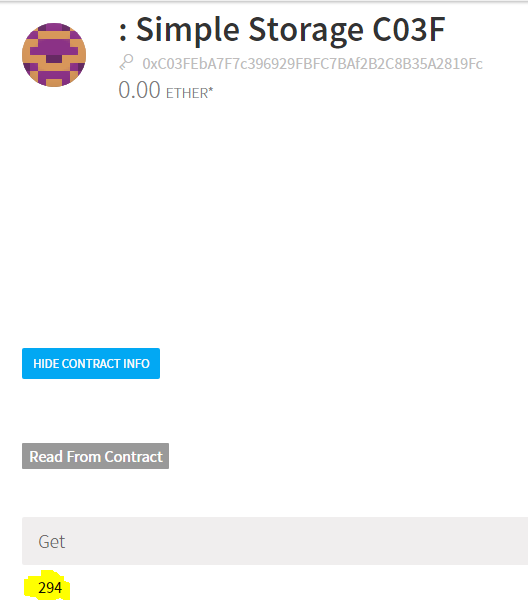
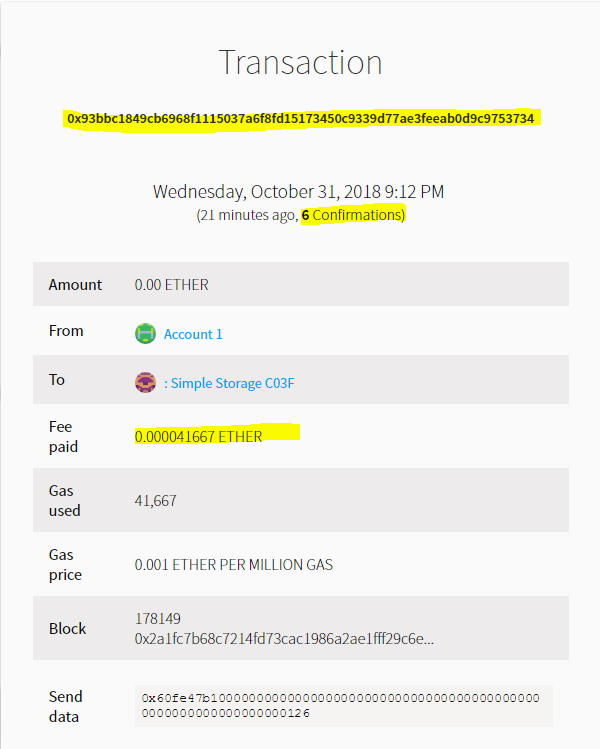
**3-4) Call functions included in the smart contract (Send a transaction)**

> Once the smart contract was written to the blockchain, it was possible to call the functions/ send transactions via the ethereum wallet interface-



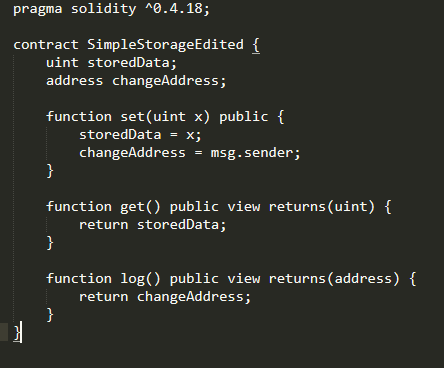


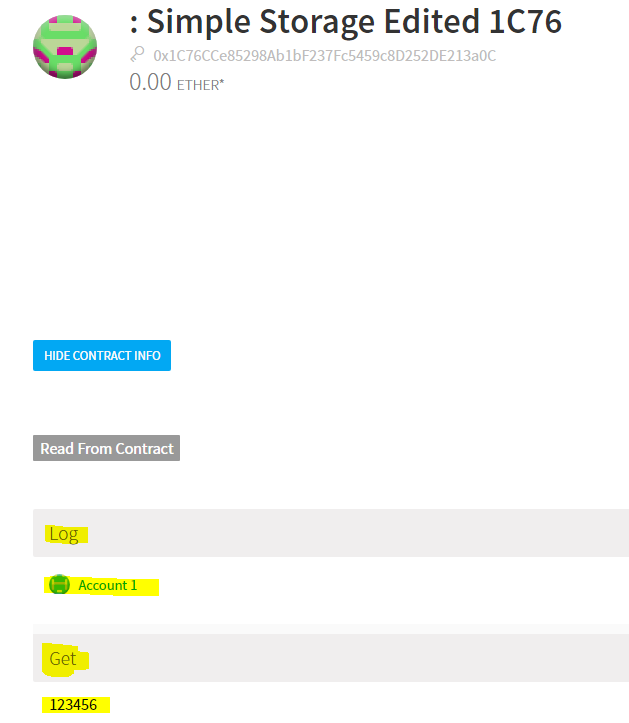
Once the transaction received sufficient confirmations, the value of the uint inside our contract- storedData is updated in the contract as follows-

**3-5) Add logging function to keep track of who recently changed data (Then, test it)**

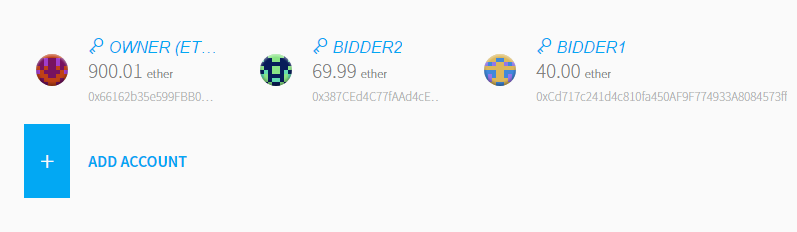
> A new solidity class file was created with the addition of the log function and deployed as a new contract and the functions were executed.





**<Step. 4> Create a smart contract for auction**

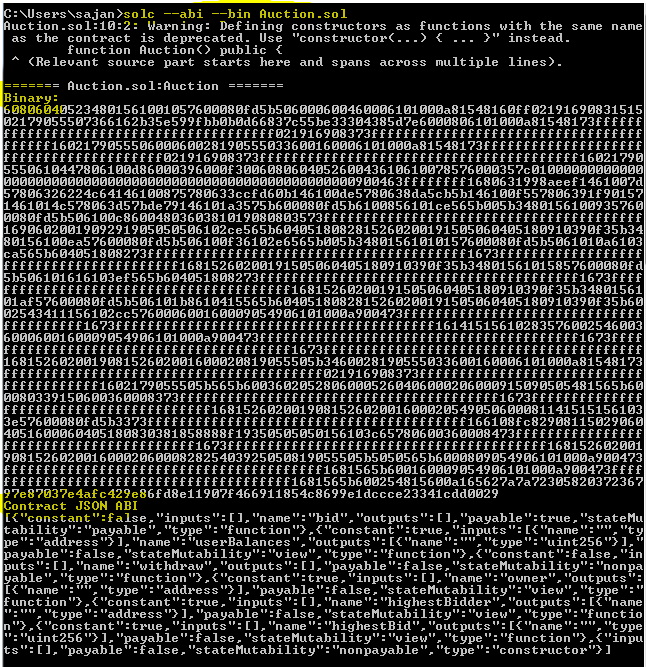
**4-0) Prepare three accounts for testing your contract. (Account 1: Owner, Account 2: Bidder 1, Account 3: Bidder 2)**



**4-1) Complete Auction smart contract and save it as Auction.sol**

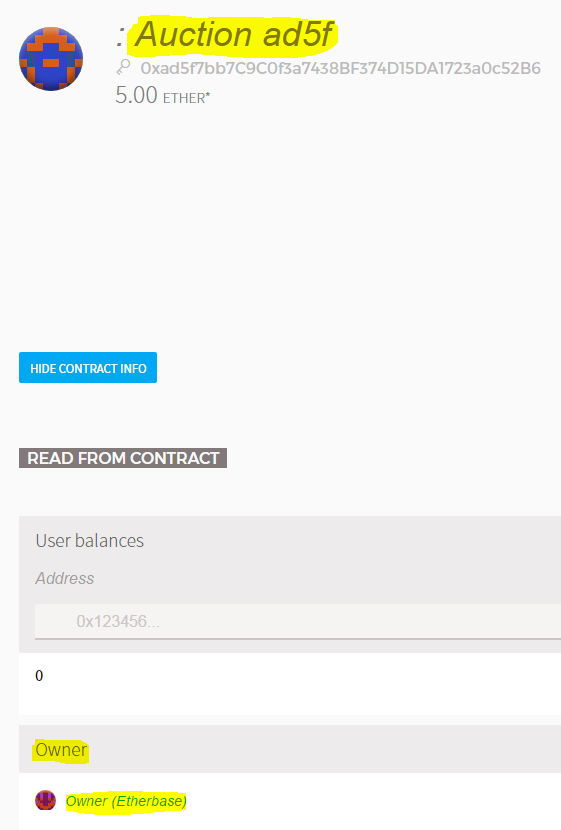
|  |
| --- |
| pragma solidity ^0.4.18;  contract Auction {  address public owner; // address of the owner who can withdraw from the auction sale  address public highestBidder; //the highest bidder's address  uint public highestBid; //the amount of the highest bid  mapping(address => uint) public userBalances; //mapping for the amount to return  bool sold;  function Auction() public {  sold = false;  owner = 0x66162b35e599FBB0B0D66837C55be33304385D7e;  highestBid = 0; //initialize highest bid with value 0  highestBidder = msg.sender;  }  function bid() public payable {  if(msg.value > highestBid) {  if(highestBidder != 0) {  userBalances[highestBidder] = highestBid;  }  highestBid = msg.value;  highestBidder = msg.sender;  }    }  function withdraw() public {  // function to withdraw the amount of bid to return  // 1. check if the amount to return is greater than zero  // 2. update status variable and return bid  address withdrawalAccount = msg.sender;  uint withdrawalAmt = userBalances[withdrawalAccount];  require(withdrawalAmt != 0);  //send the amount back  if(msg.sender.send(withdrawalAmt)) {  userBalances[withdrawalAccount] -= withdrawalAmt;  }  }  } |

**4-2) Compile the code using solc. (After installing solc, use solc –abi –bin)**



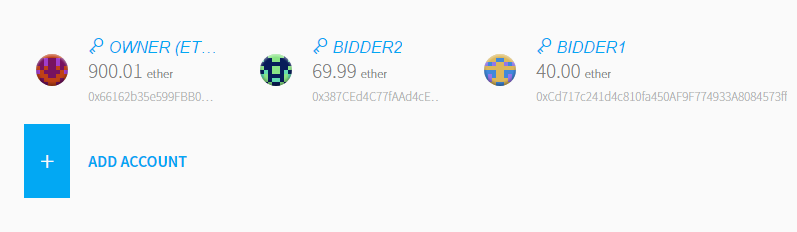
**4-3) Send transaction to create the contract on account 1 (owner)**

> As seen from step 4-0, 3 different accounts had been created prior. The contract was created using ethereum wallet by owner account



**4-4) Check the balance of contract account and account 1 and account 2, the amount of highest bid and the address of the higest bidder**

> The pre-existing balance of owner, account 1 and account 2 were as follows-

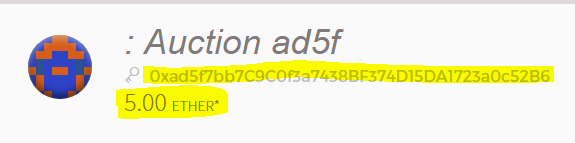


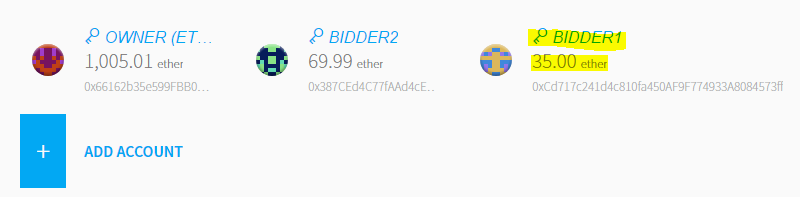
Highest bid was initialized to 0 and highest bidder was initialized to *msg.sender* in the constructor

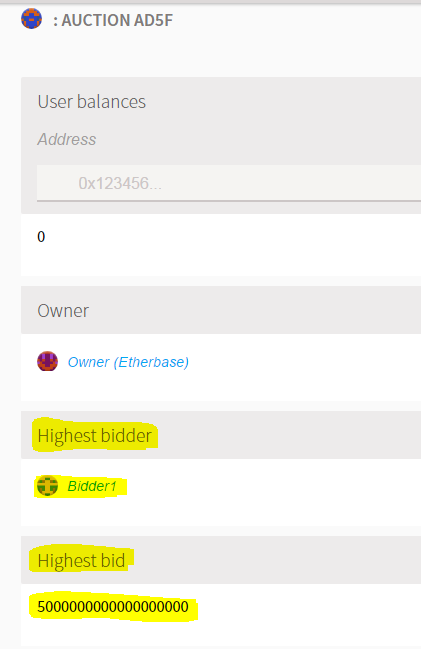
**4-5) Send transaction to execute bid() function in order to bid 5ETH, on account 2(Bidder 1)**

> Bidder 1 put a bid() with 5ETH using the ethereum wallet

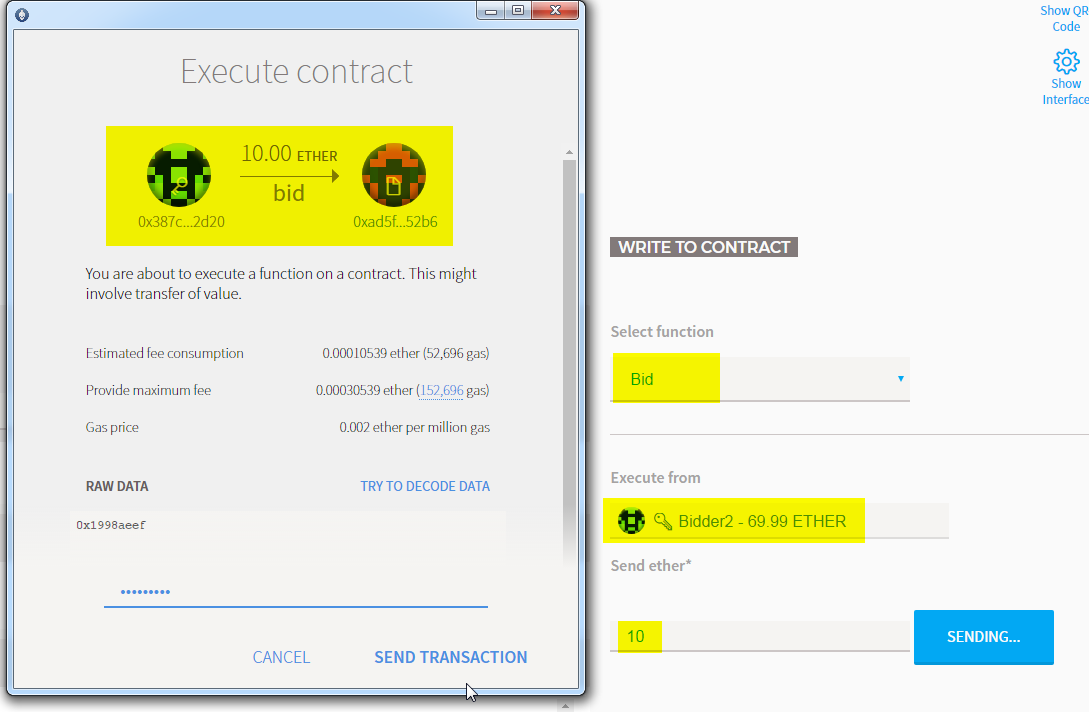
4-6) Check the balance of the contract account, account 2 and account 3, the amount of highest bid and the address of the highest bidder





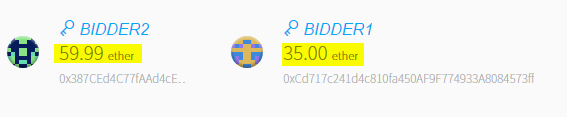


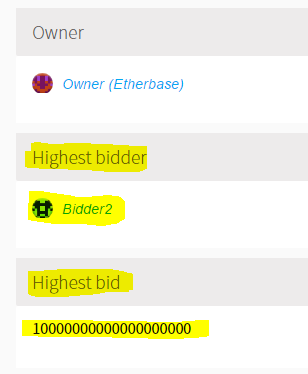
**4-7) Send transaction to execute bid() function in order to bid 10ETH, on account 3(bidder 2).**



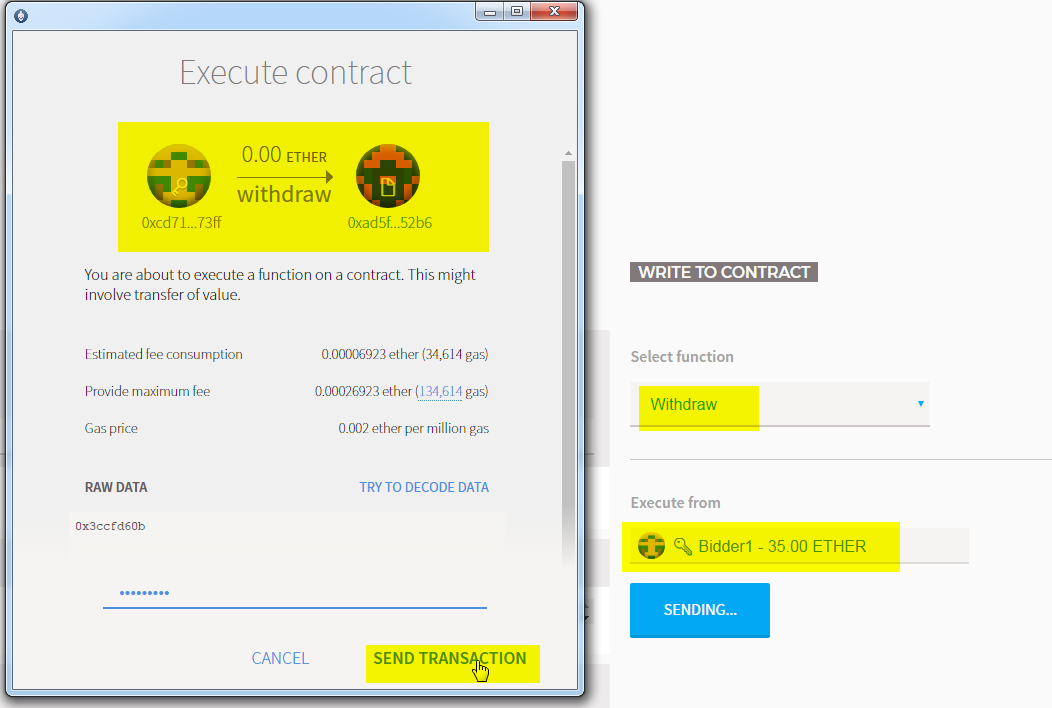
**4-8) Check the balance of contract account and account 3, the amount of the highest bid and the address of the highest bidder**



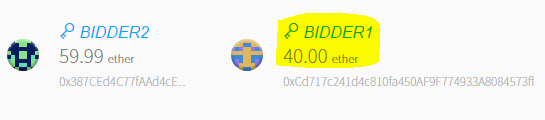


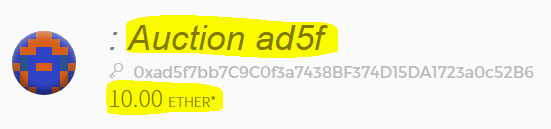


**4-9) Send transaction to execute withdraw() function in order to withdraw the amount of bid, on account 2 (bidder 1)**



**4-10) Check the balance of contract account and account 2(bidder 1)**

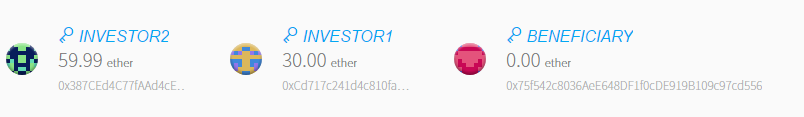




**<Step. 5> Create a smart contract for crowd funding**

**5-0) Prepare three accounts for testing your contract. (Account 1: Beneficiary, Account 2: Investor1, Account 3: Investor2)**

> The three accounts were prepared as-

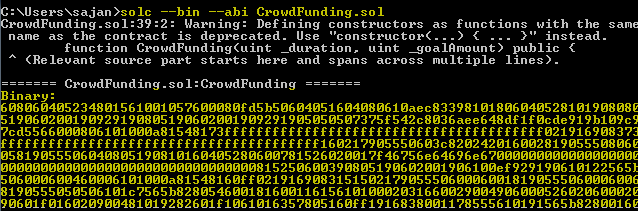


**5-1) Complete crowdfunding smart contract and save it as “CrowdFunding.sol”**

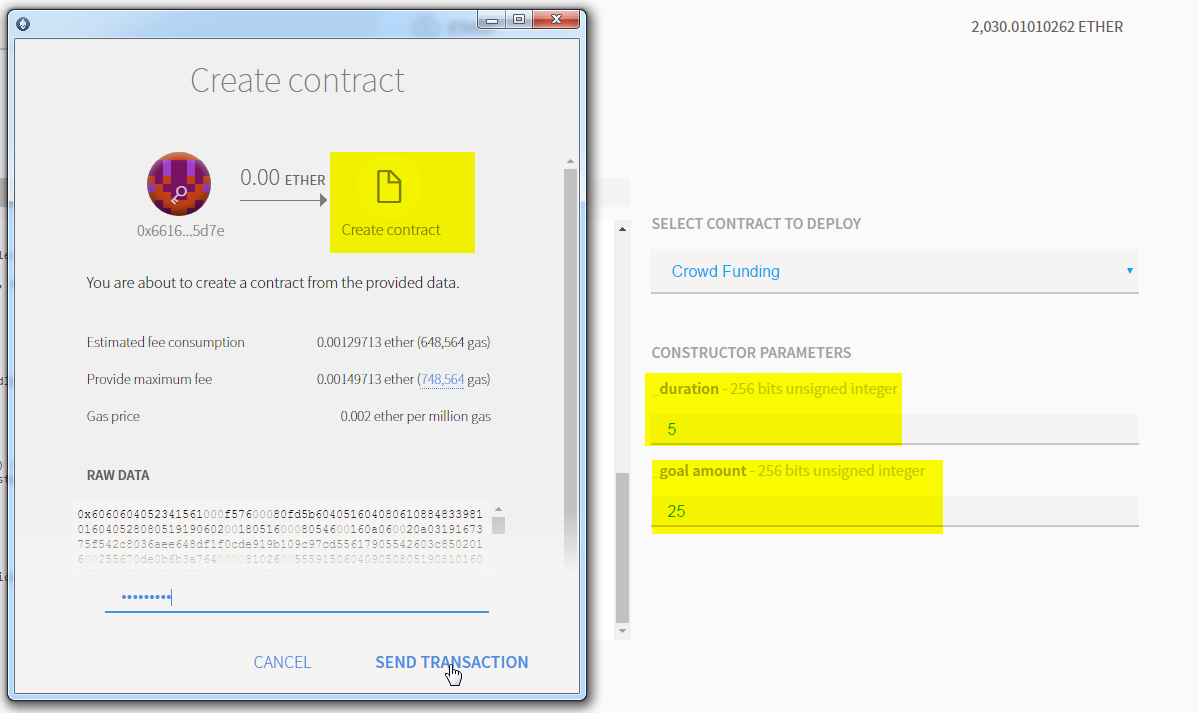
|  |
| --- |
| pragma solidity ^0.4.18;  contract CrowdFunding {  // Investor struct  struct Investor {  address addr; // investor's address  uint amount; // investment amount  }  address public beneficiary; // contract beneficiary  uint public numInvestors; // the number of investors  uint public deadline; // deadline for this contract to be closed  string public status; // "Funding", "Campagin Success", "Campagin Failed"  bool public end; // the end of funding  uint public goalAmount; // target amount  uint public totalAmount; //total ammout  Investor[] public investors; //declare an array of structs of Investors to store records  // 1. Create modifier to limit to beneficiary  modifier onlyBeneficiary() {  require(msg.sender == beneficiary);  \_;  }  modifier beforeDeadline() {  require(now < deadline);  \_;  }  modifier afterDeadline() {  require(now >= deadline);  \_;  }  modifier notEnded() {  require(end == false);  \_;  }    // Constructor  function CrowdFunding(uint \_duration, uint \_goalAmount) public {  beneficiary = 0x75f542c8036AeE648DF1f0cDE919B109c97cd556;  deadline = now + \_duration \* 1 minutes;  goalAmount = \_goalAmount \* 1e18;  status = "Funding";  end = false;  numInvestors = 0;  totalAmount = 0;  }  // Function to be called when investing  function fund() public beforeDeadline payable notEnded {  uint amt = msg.value;  Investor memory i = Investor(msg.sender, amt);  investors.push(i);  totalAmount += amt;  numInvestors++;  }  function checkGoalReached() public afterDeadline {  if(totalAmount >= goalAmount) {  beneficiary.transfer(totalAmount);  status = "Campaign Success";  }  else {  for(uint k=0; k < numInvestors; k++) {  investors[k].addr.transfer(investors[k].amount);  }  status = "Campagin Failed";  }  end = true;  }  function destroyContract() public onlyBeneficiary {  selfdestruct(beneficiary);  }  } |

**5-2) Compile the code using solc**

(You can use ***solc –abi –bin CrowdFunding.sol*** command to check)

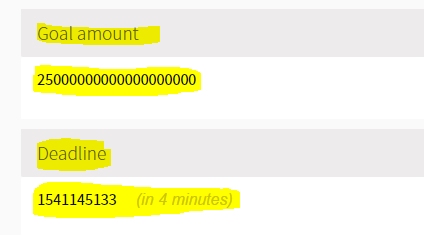


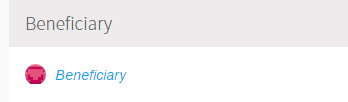
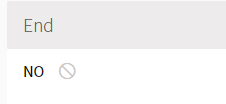
**5-3) Send transaction to create contract (When creating contract, set duration to 5mins and goal amount to 25ETH)**

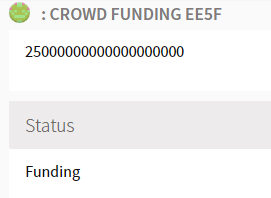


**5-4) Check deadline and goal amount and end state**

> We here consider a latency of 1 minute (while taking snapshot and writing the contract in the blockchain) causing remaining time to be 4 mins as shown below. The Goal amount is shown in Wei units which is 1 ETH = 10^18 Wei.

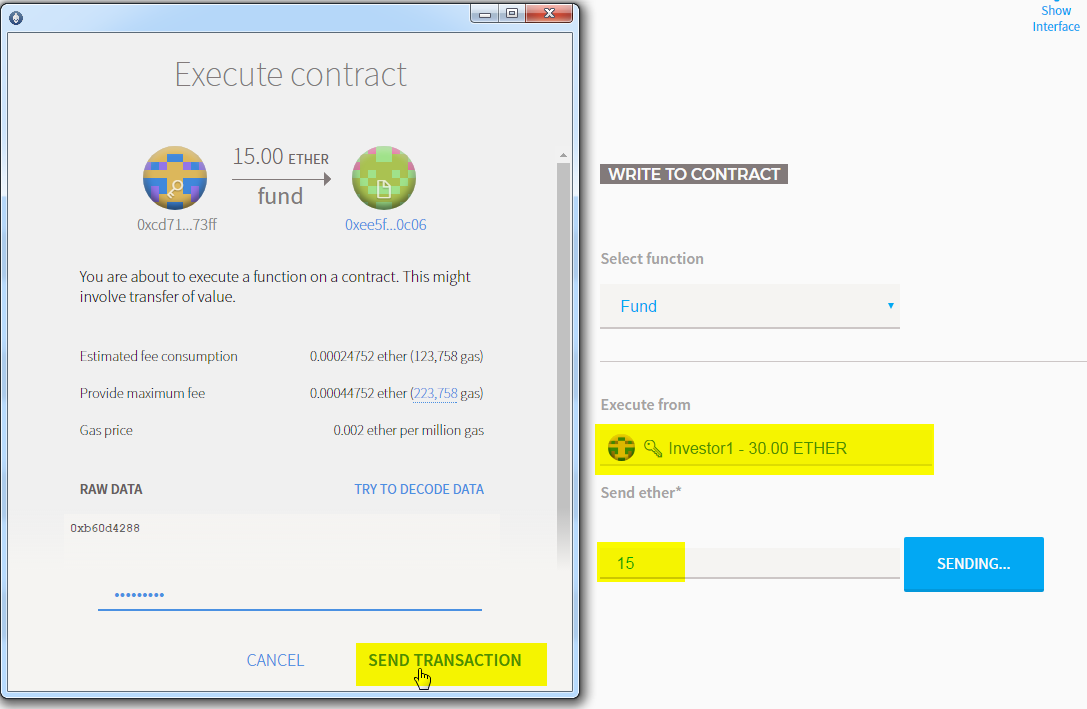


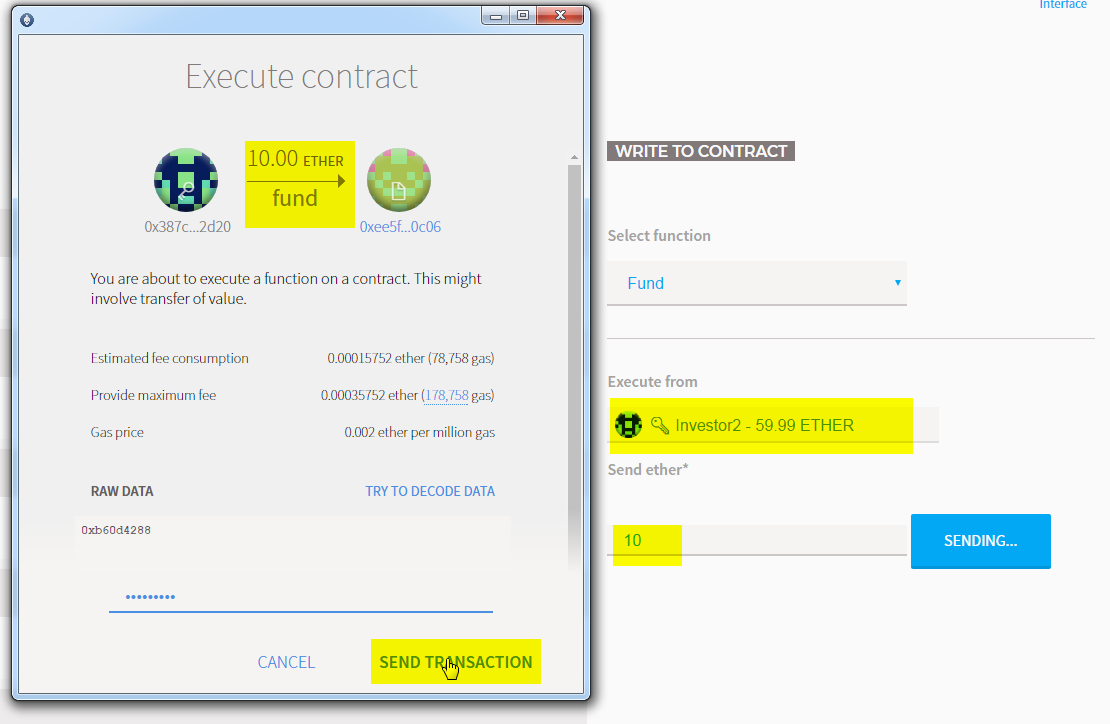
 

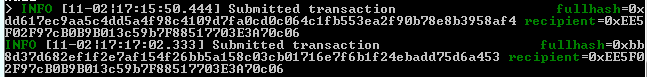


**<<Case of Successful Funding>>**

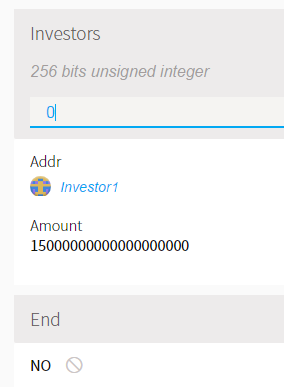
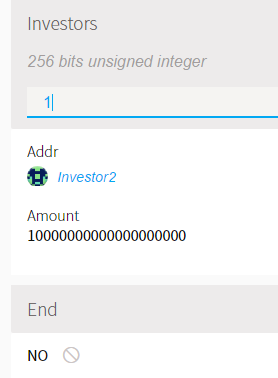
**5-5) Send transaction to execute fund() function in order to fund 15ETH and 10ETH from Investor 1 and Investor 2 respectively**



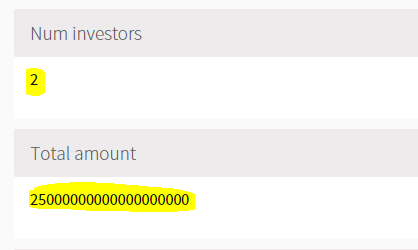




**5-6) Check investment of Investor 1 and 2**

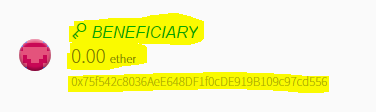
 

**5-7) Check total investment of the contract and the balance of the contract account**

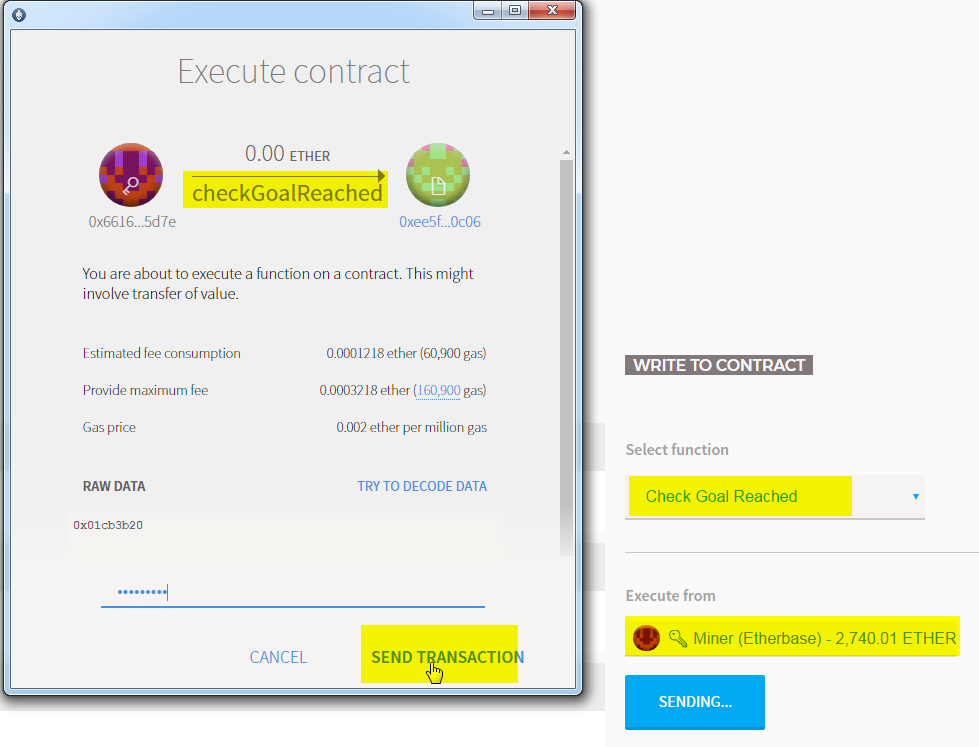
 

**5-8) Check Beneficiary’s account**

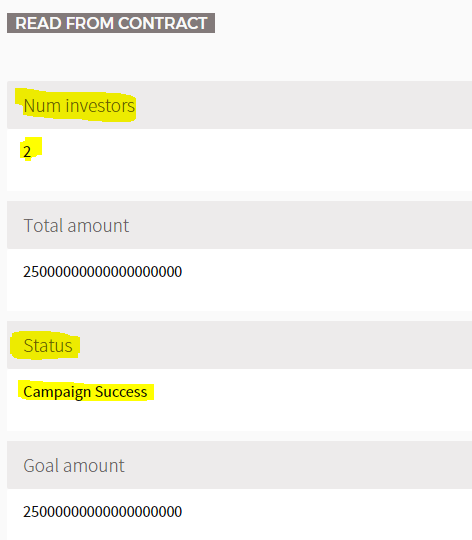
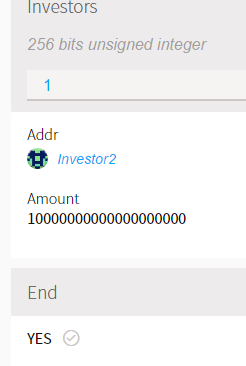
> We initialized the Beneficiary’s account with balance 0 in the first place and we can check the beneficiary’s account balance from ethereum wallet GUI as-



**5-9) Send transaction to execute *checkGoalReached()* function to check the fundraising results (After deadline)**

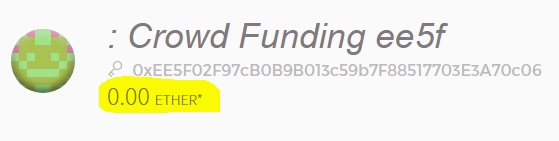


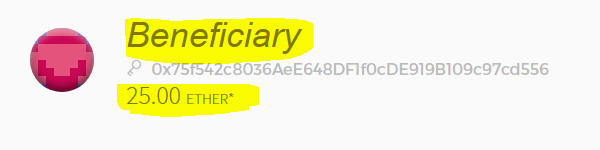
**5-10) Check deadline and end state**

**5-11) Check balance of contract account and owner**

> On success of crowdfunding, the collected fund is moved from contract account to the beneficiary account as shown below-

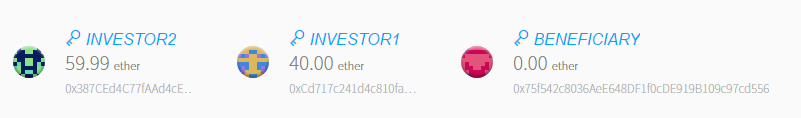


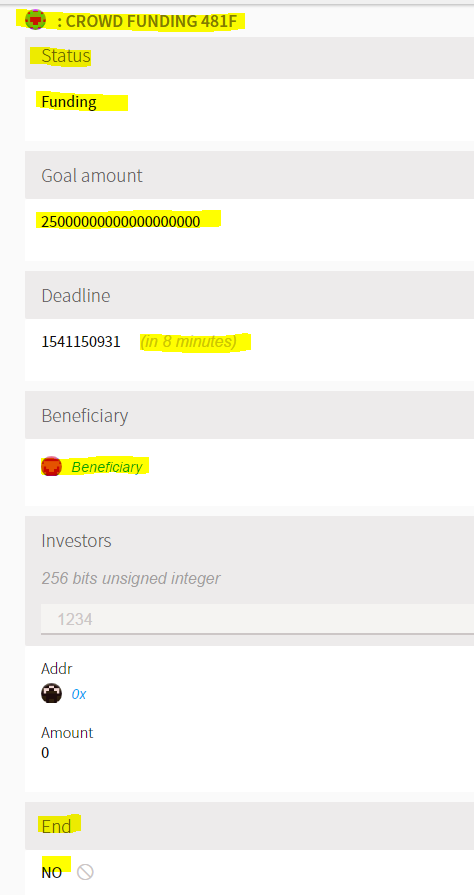


**<<Case of failed funding>>**

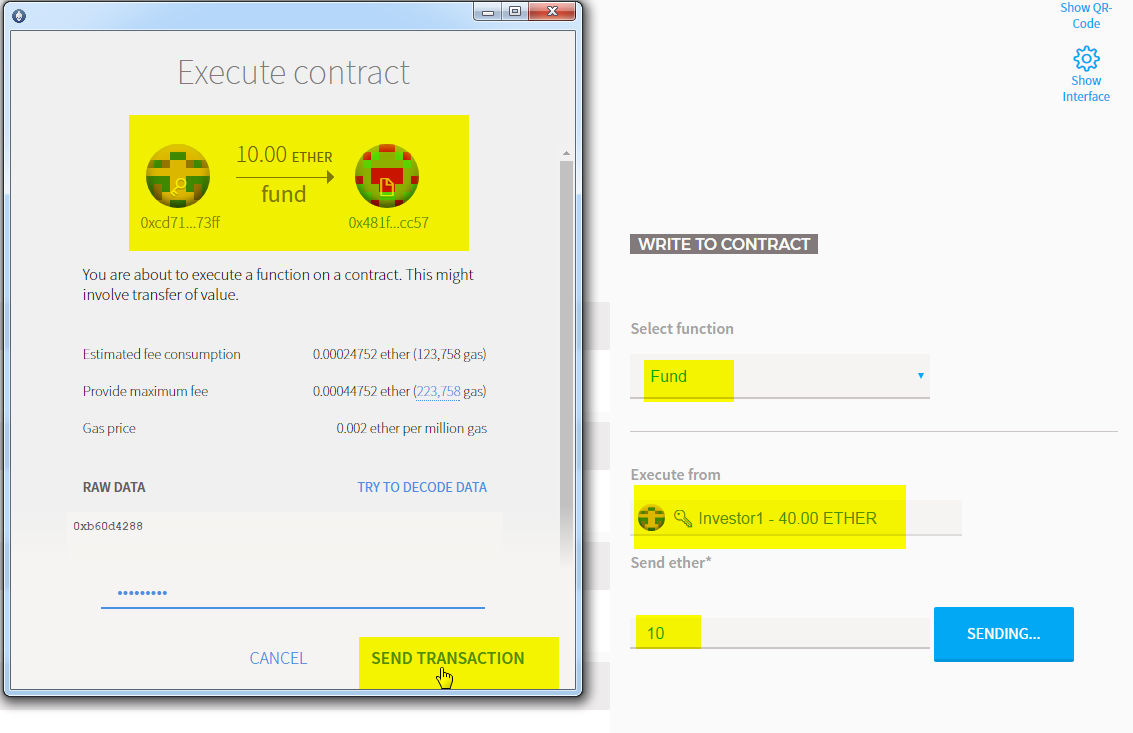
**5-12) Follow steps from 5-3) and 5-4)**

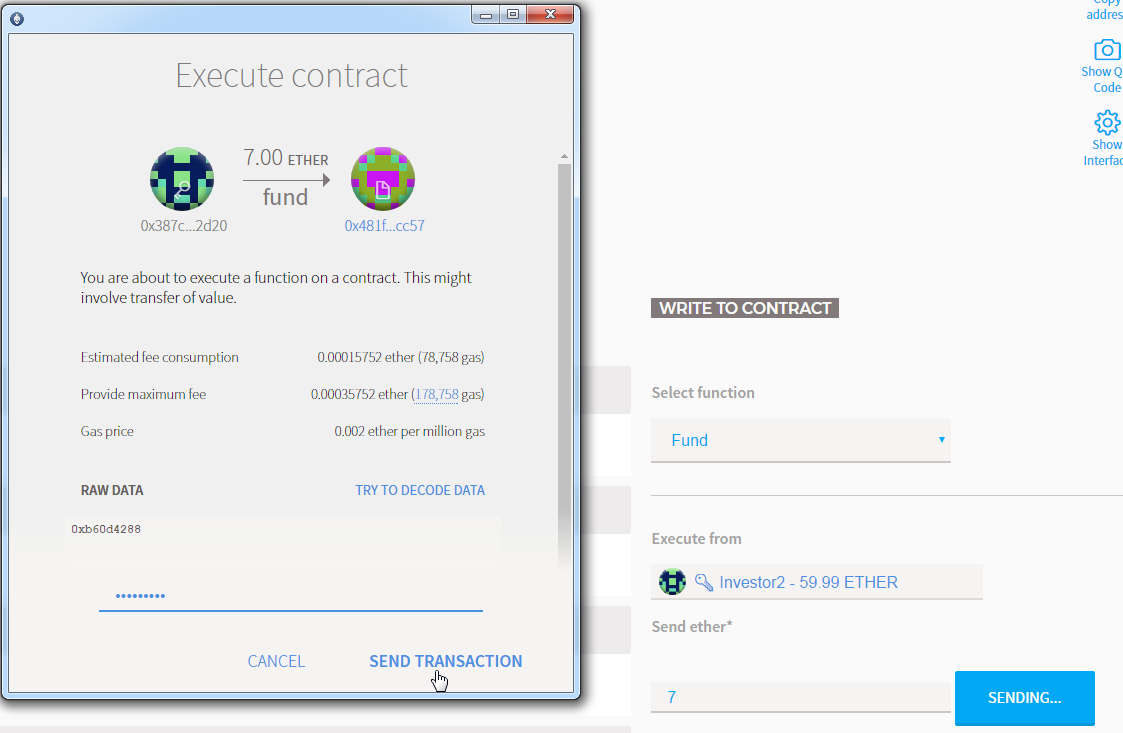
> A new contract was created with the goal amount of 25ETH and a deadline in the next 10 minutes and we ***re-started*** with the following balances for investor 1, investor 2 and beneficiary



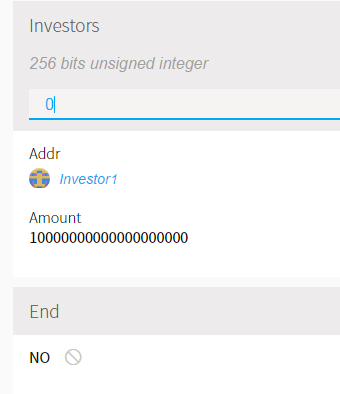
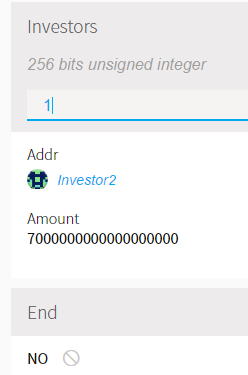


**5-13) Send transaction to execute fund() function in order to fund 10ETH from 7ETH from Investor 1 and Investor 2 respectively**

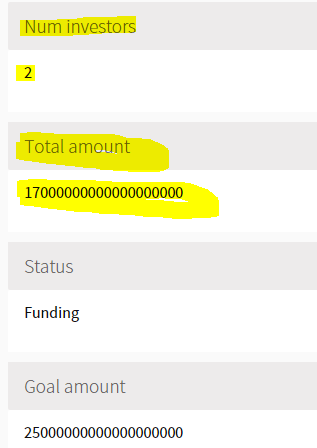


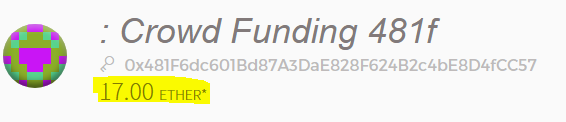


**5-14) Check investment of investor 1 and 2**

**5-15) Check total investment and balance of the contract account**



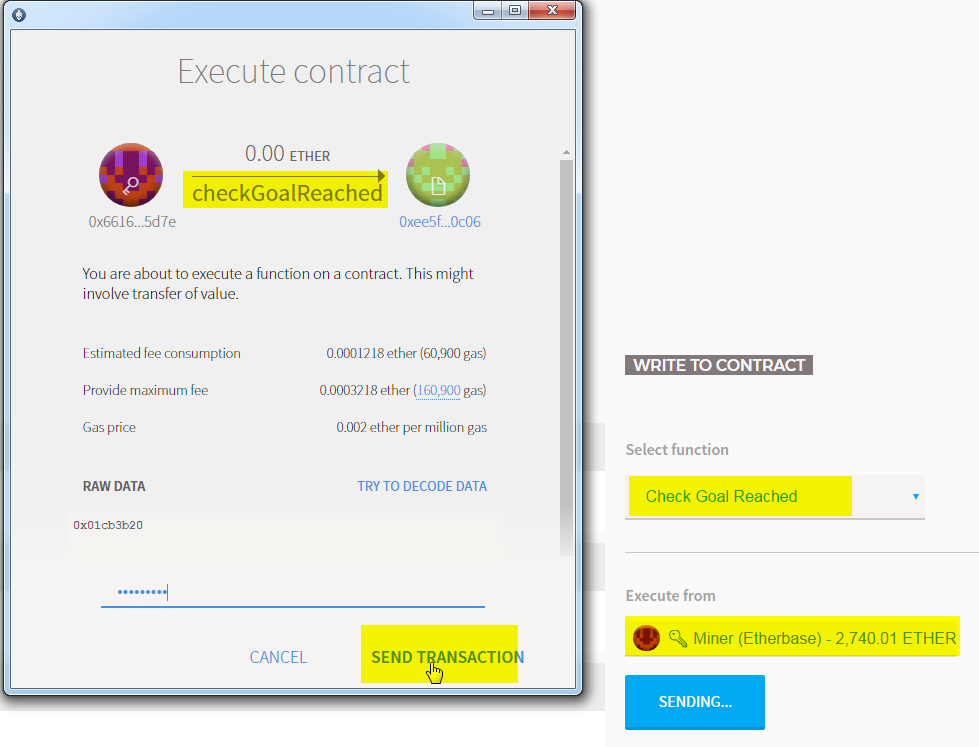


**5-16) Check balance of investor 1, investor 2 and beneficiary account**



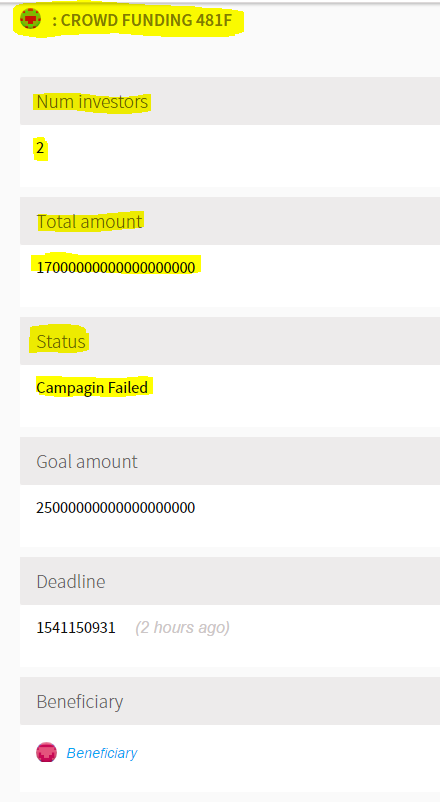
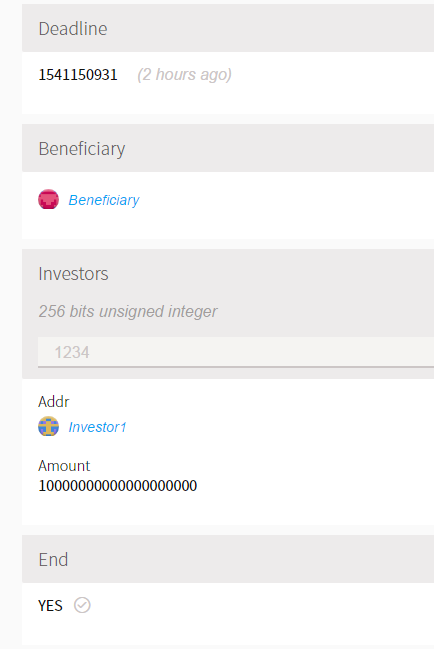
**5-17) Send transaction to execute checkGoalReached() function to check the fundraising results (After deadline)**

> checkGoalReached function was executed via the GUI interface of etheruem wallet. The function being submitted to the blockchain can be seen from the snapshot below-

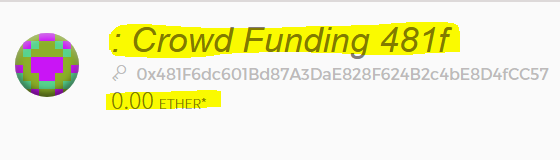




**5-18) Check deadline and end state**

**5-19) Check balance of contract account**



**5-20) Check balance of investor 1, investor 2 and beneficiary account**

> After the failure of funding, the collected funds were re-distributed to the original investors automatically, which can be confirmed by the snapshot below-

