Auction Dapp Part1

Now that you have a basic idea of what Ethereum and Dapps are, we can build our auction Dapp. An auction is typical example, but it is complex enough to provide a perfect Dapp.

To summarize, our auction Dapp will be a web application that enables users to start in auctions using ether. Lets see what we can throw together:

1. Setting up the development environment
2. Creating a Truffle project
3. Writing the smart contract
4. Compiling and migrating the smart contract on geth private chain using truffle
5. Interact with our contract through Express Node JS based web app

- Prerequisites

Before we start we need some tools and dependencies. Please install the following:

- Node.js and npm (comes with Node)
- Git
- Geth

Once we have those installed, we need to install truffle to create a truffle project. Install using the following command.

```
$ npm install -g truffle
```

Next, we need to install express-generator to create an express application template. Install using the following command.

```
$ npm install express-generator -g
```

Auction Description

A vehicle’s owner deploys the contract to the blockchain and becomes the auction owner. The auction is open immediately after the contract deployment, and, once the bidding period is over, the highest bidder wins the auction, and the other participants get back their bids.

Auction Contract in Solidity

To write our solution contract, we will use solidity,

- **Problem Statement**

Implement a smart contract for Auction.

- The contract should have exactly one owner.
- Clients can bid vehicle for time slots provided by owner
- once the bidding period is over, the highest bidder wins the auction, and the other participants get back their bids.
- once the bidding period is over, the highest bidder wins the auction, and contract owner gets highest bid

Setting up the development environment with Truffle

Step1: Creating and initializing project

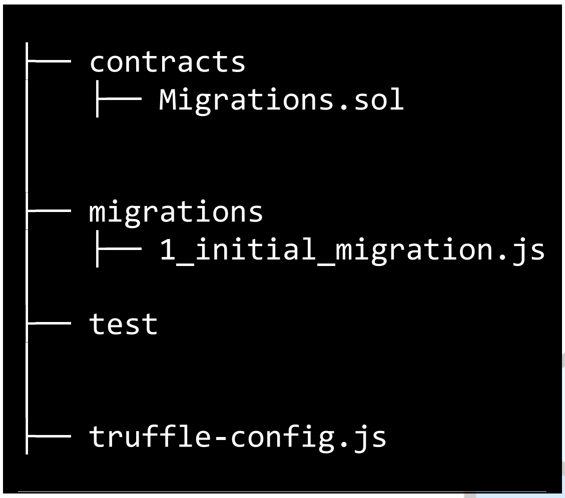
```
$ mkdir <NameOfMyProject>
$ cd <NameOfMyProject>
$ truffle init
```

___Truffle Output____

```
Downloads...
Unpacking...
Setting up...
Unbox successful. Sweet!

Commands:
Compile:
Migrate:
Test contracts:
truffle compile
truffle migrate
truffle test
```

Truffle will create a set of folders: “contracts/,” “migrations/,” “test/,” and a truffle configuration file, “truffle-config.js.”

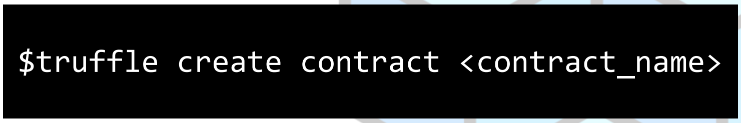
Your Truffle folder should looks like this:



```
├── contracts
│   ├── Migrations.sol
│
│
├── migrations
│   ├── 1_initial_migration.js
│
├── test
│
│
├── truffle-config.js
```

Now you can change this folder structure with your contract requirements

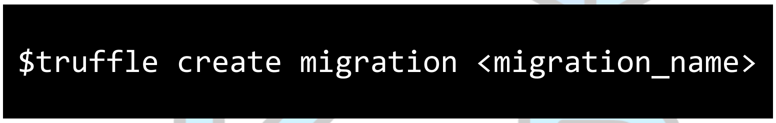
To create a new contract file under contracts folder , use following command



```
$truffle create contract <contract_name>
```

Example: **truffle create contract Auction**

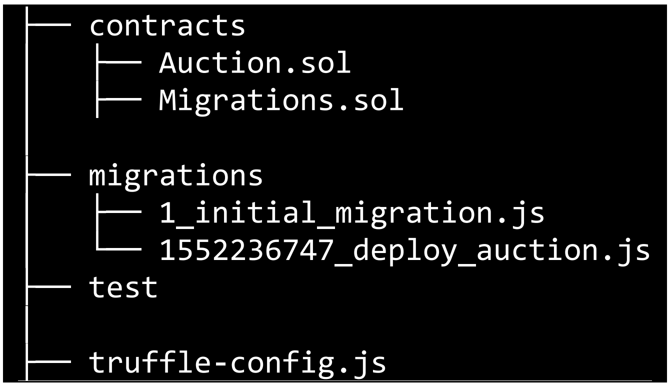
To configure deployment of new contract , create a script file under migration folder for this use following command



```
$truffle create migration <migration_name>
```

Example: **truffle create migration deploy_auction**

Now your Truffle folder should look like this



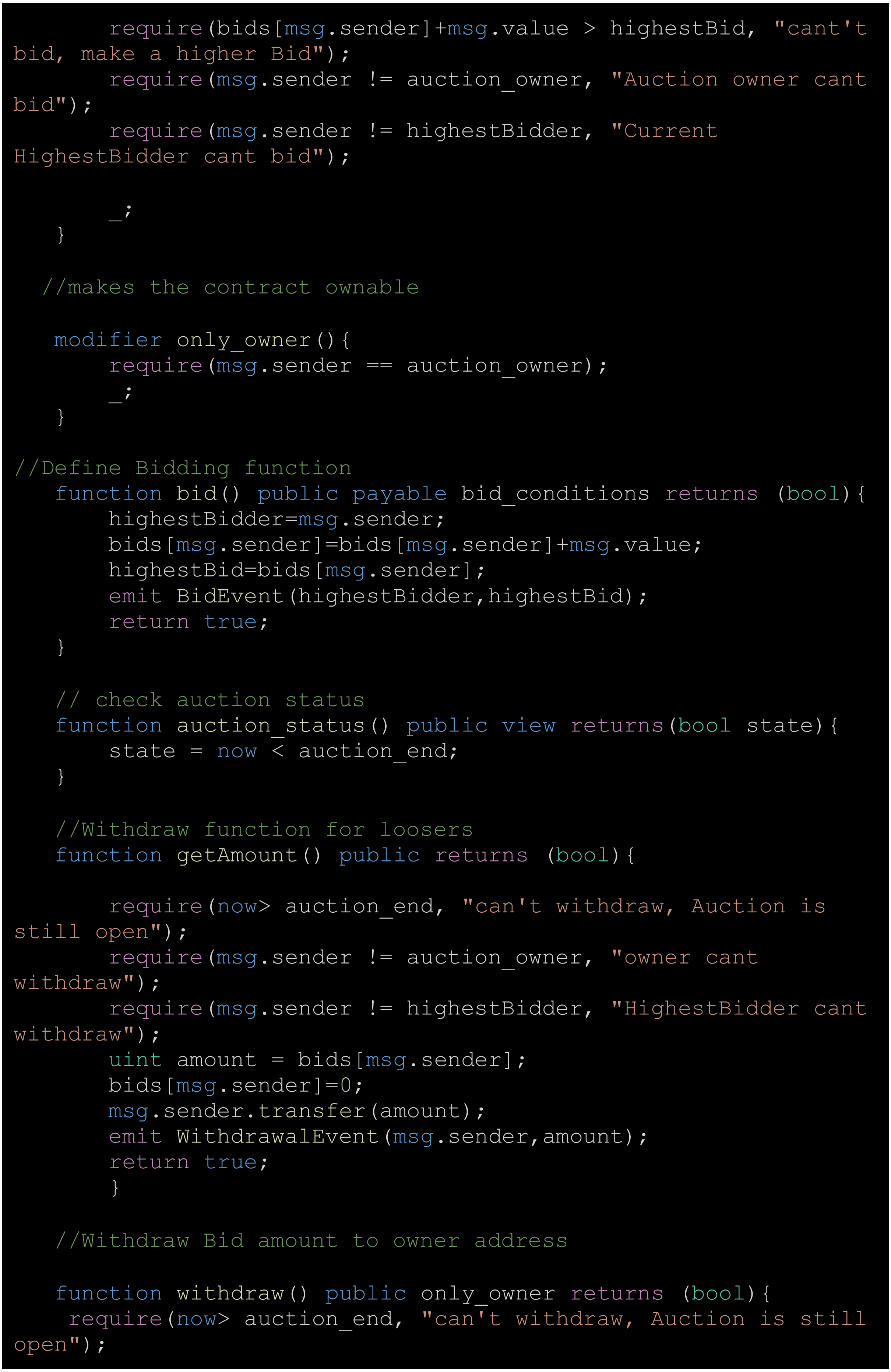
```
├── contracts
│   ├── Auction.sol
│   ├── Migrations.sol
│
├── migrations
│   ├── 1_initial_migration.js
│   └── 1552236747_deploy_auction.js
├── test
│
├── truffle-config.js
```

Contract Skeleton

```
pragma solidity ^0.5.0;
contract Auction{
   //Declare All State Variables here

   //Define a constructor for your contract
   constructor()public {

}



//Define a structure for Vehicle Details
   struct car{
       string Brand;
       string Rnumber;
   }
   car public Mycar;
   //Create a dynamic array to contain all of the bidders
address
   address[] bidders;

   //Mapping that accepts the bidder's address as the key, and
with the value type being the corresponding bid

   mapping(address => uint) public bids;
   //Checks whether the bid is can be done
   modifier bid_conditions(){

   }

   //makes the contract ownable
   modifier only_owner(){

   }
//Define Bidding function
   function bid() public payable bid_conditions returns (bool){

   }
 //Withdraw function for loosers
   function getAmount() public returns (bool){
             }

  //Withdraw Bid amount to owner address

   function withdraw() public only_owner returns (bool){


   }
```

Step2: Solution for Auction Contract

```
pragma solidity ^0.5.0;

contract Auction{
   //Declare All State Variables here
   address internal auction_owner;
   uint256 public auction_end;
   uint256 public highestBid;
   address payable public highestBidder;
   //Define a constructor for your contract
   constructor(uint _biddingTime, string  memory _brand, string
memory _Rnumber) public {

       auction_owner=msg.sender;
       auction_end =now +_biddingTime * 1 minutes;
       Mycar.Brand =_brand;
       Mycar.Rnumber =_Rnumber;
       Mycar.owner = auction_owner;

   }

//Function for get auction details

    function getAuctionDetails() public view returns
(uint256,uint256,address,address) {
      return
(auction_end,highestBid,highestBidder,auction_owner);
  }

   //Define a structure for Vehicle Details
   struct car{
       string Brand;
       string Rnumber;
       address owner;
   }
   car public Mycar;

   //Mapping that accepts the bidder's address as the key, and
with the value type being the corresponding bid
   mapping(address => uint) public bids;
   event BidEvent (address indexed highestBidder,uint256
highestBid);
   event WithdrawalEvent(address withdrawer,uint256 amount);
   //Checks whether the bid is can be done
   modifier bid_conditions(){

       require(now<= auction_end,"auction timeout");
```



```
       require(bids[msg.sender]+msg.value > highestBid, "cant't
bid, make a higher Bid");
       require(msg.sender != auction_owner, "Auction owner cant
bid");
       require(msg.sender != highestBidder, "Current
HighestBidder cant bid");

       _;
   }

  //makes the contract ownable

   modifier only_owner(){
       require(msg.sender == auction_owner);
       _;
   }

//Define Bidding function
   function bid() public payable bid_conditions returns (bool){
       highestBidder=msg.sender;
       bids[msg.sender]=bids[msg.sender]+msg.value;
       highestBid=bids[msg.sender];
       emit BidEvent(highestBidder,highestBid);
       return true;
   }

   // check auction status
   function auction_status() public view returns(bool state){
       state = now < auction_end;
   }

   //Withdraw function for loosers
   function getAmount() public returns (bool){

       require(now> auction_end, "can't withdraw, Auction is
still open");
       require(msg.sender != auction_owner, "owner cant
withdraw");
       require(msg.sender != highestBidder, "HighestBidder cant
withdraw");
       uint amount = bids[msg.sender];
       bids[msg.sender]=0;
       msg.sender.transfer(amount);
       emit WithdrawalEvent(msg.sender,amount);
       return true;
       }

   //Withdraw Bid amount to owner address

   function withdraw() public only_owner returns (bool){
    require(now> auction_end, "can't withdraw, Auction is still
open");
```

```
        msg.sender.transfer(highestBid);
        Mycar.owner = highestBidder;
        emit WithdrawalEvent(msg.sender,highestBid);
        return true;
  }
}
```

Step3: Network configuration in Truffle

Truffle also requires that you have a running Ethereum client which supports the standard JSON RPC API , Use geth private chain as Ethereum client
Edit the “truffle-config.js” file to:

```
module.exports = {
 development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 8545,            // Standard Ethereum port (default:
none)
   network_id: "4002",       // Any network (default: none)
   gas:6283185

  },
  };
```

Step4: Geth Private Chain with 5 Accounts

Create a directory and open terminal with in the directory and use following command

```
$ geth --identity "miner" --networkid 4002 --datadir . --rpc --
rpcport "8545" --unlock 0 --ipcpath "~/.ethereum/geth.ipc" --
rpccorsdomain "*" --rpcapi "db,eth,net,web3,personal" --dev
```

In another terminal call: geth attach
Use the below command to create 5 accounts with password as blank

```
>personal.newAccount("")
>personal.newAccount("")
>personal.newAccount("")
```

```
>personal.newAccount("")
>personal.newAccount("")
```

Use the below command to transfer ether from coinbase, to other accounts

```
>eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[1],valu
e:50000000000000000000})
>eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[2],valu
e:50000000000000000000})
>eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[3],valu
e:50000000000000000000})
>eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[4],valu
e:50000000000000000000})
>eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[5],valu
e:50000000000000000000})
```

Step4: Deployment configuration in Truffle

Configure deployment script **2_deploy_auction.js**, use following script file

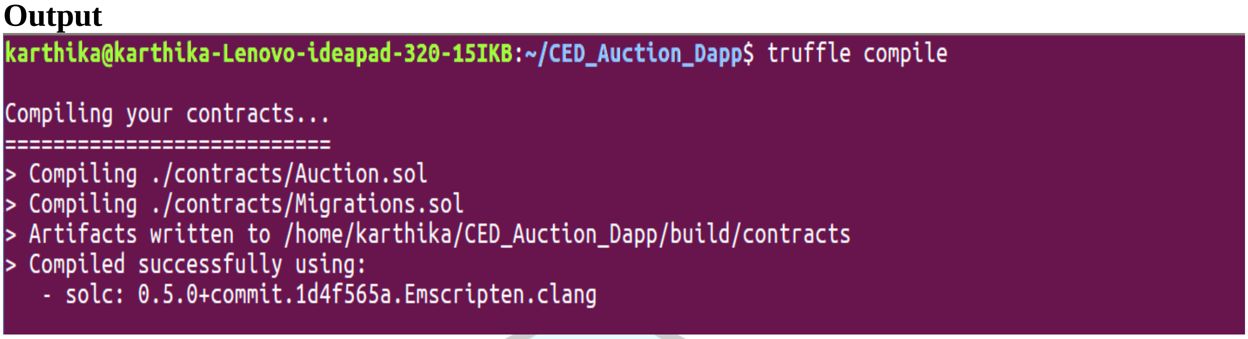
```
const Auction = artifacts.require("Auction");

module.exports = function(deployer) {
//we need to pass constructor parameters here at the time of
deployment
 deployer.deploy(Auction,60,"BMW","RN00091");
};
```

Step5: Compiling Contracts

To compile a Truffle project, change to the root of the directory where the project is located and then type the following into a terminal:

```
$ truffle compile
```

Output



```
karthika@karthika-Lenovo-ideapad-320-15IKB:~/CED_Auction_Dapp$ truffle compile

Compiling your contracts...
===========================
> Compiling ./contracts/Auction.sol
> Compiling ./contracts/Migrations.sol
> Artifacts written to /home/karthika/CED_Auction_Dapp/build/contracts
> Compiled successfully using:
   - solc: 0.5.0+commit.1d4f565a.Emscripten.clang
```

Artifacts of your compilation will be placed in the build/contracts/ directory, relative to your project root.

Step6: Running Migration with Truffle

Migrations are JavaScript files that help you deploy contracts to the Ethereum network. Here we have Geth as Ethereum network. So we need to run geth private chain in your machine.

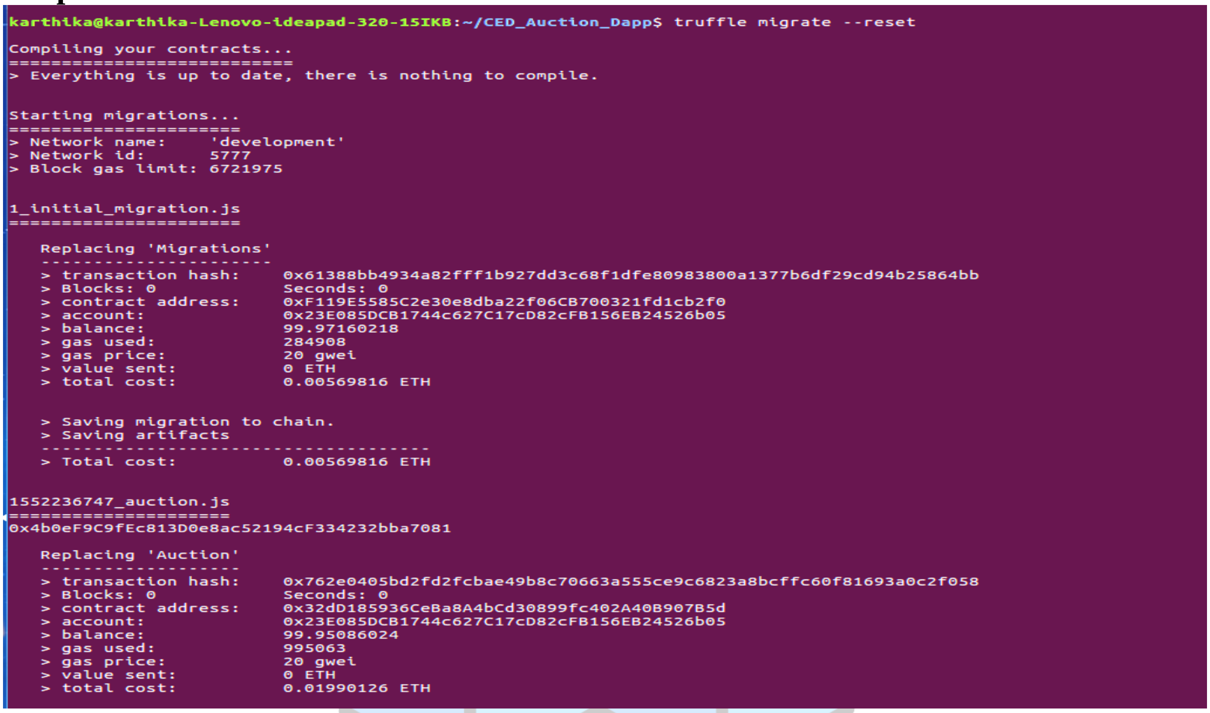
To run your migrations, run the following:

```
$ truffle migrate
```

This will run all migrations located within your project's **migrations** directory. At their simplest, migrations are simply a set of managed deployment scripts. If your migrations were previously run successfully, **truffle migrate** will start execution from the last migration that was run, running only newly created migrations. If no new migrations exists, **truffle migrate** won't perform any action at all. You can use the **--reset** option to run all your migrations from the beginning. For local testing make sure to have a test blockchain such as Ganache installed and running before executing **migrate**.

You can see the migrations being executed in order, followed by some information related to each migration.

Output



```
karthika@karthika-Lenovo-ideapad-320-15IKB:~/CED_Auction_Dapp$ truffle migrate --reset

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0x61388bb4934a82fff1b927dd3c68f1dfe80983800a1377b6df29cd94b25864bb
   > Blocks: 0            Seconds: 0
   > contract address:    0xF119E5585C2e30e8dba22f06CB700321fd1cb2f0
   > account:             0x23E085DCB1744c627C17cD82cFB156EB24526b05
   > balance:             99.97160218
   > gas used:            284908
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00569816 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00569816 ETH


1552236747_auction.js
=====================
0x4b0eF9C9fEc813D0e8ac52194cF334232bba7081

   Replacing 'Auction'
   -------------------
   > transaction hash:    0x762e0405bd2fd2fcbae49b8c70663a555ce9c6823a8bcffc60f81693a0c2f058
   > Blocks: 0            Seconds: 0
   > contract address:    0x32dD185936CeBa8A4bCd30899fc402A40B907B5d
   > account:             0x23E085DCB1744c627C17cD82cFB156EB24526b05
   > balance:             99.95086024
   > gas used:            995063
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01990126 ETH
```

Check your Running **geth** , here list all transaction logs.