Practical-9

Name: Dhruv Sonani

Roll No.: 20BCE527

Course Name and Code: Blockchain Technology(2CSDE93)

Aim: To write a Solidity contract that implements a distributed ticket sales system. Anybody can create an event (specifying the initial price and number of tickets). Anybody can then purchase one of the initial tickets or sell those tickets peer-to-peer. At the event, gate agents will check that each attendee is listed in the final attendees list on the blockchain.

Code:

```
pragma solidity >=0.7.0 < 0.9.0;

contract TicketSystem{

    address payable public owner;

    constructor() payable{
        owner = payable(msg.sender);

    }

    address[] finalAttendees;

    function uintToString(uint v) internal pure returns(string memory str){
        uint maxlength =100;
        bytes memory reversed = new bytes(maxlength);
        uint i=0;
        while(v!=0){
            uint8 remainder = uint8(v%10);
            v=v/10;
            reversed[i++] = bytes1(uint8(48 + remainder));

        }
        bytes memory s =new bytes(i);
        for(uint j=0;j<i;j++){
            s[j] = reversed[i - 1 - j];

        }
        str = string(s);
        return str;
```

```
    }

    function stringAppend(string memory str1, string memory str2)internal pure
returns(string memory str){
        bytes memory a = new bytes(bytes(str1).length);
        bytes memory b = new bytes(bytes(str2).length);
        for(uint i=0;i<a.length;i++){
            a[i] = bytes(str1)[i];
        }

        for(uint i=0;i<b.length;i++){
            b[i] = bytes(str2)[i];
        }
        bytes memory result = new bytes(a.length +b.length);
        uint j=0;
        for(uint i=0;i<a.length;i++){
            result[j] = a[i];
            j++;
        }
        for(uint i=0;i<b.length;i++){
            result[j] = b[i];
            j++;
        }
        str = string(result);
        return str;
    }

    struct TicketHolder {
        uint total_tickets;
        uint toSell;
    }

    struct Event{
        string name;
        uint total_tickets;
        uint price;
        mapping(address => TicketHolder) ticketHolders;
        address[] keys;
    }

    mapping(address=>Event) events;
    mapping(string=>uint) eventNames;
    mapping(string=>address) eventOwners;
```

```
    function checkForEventDeclaration(address _owner) internal view
returns(bool){
        if(events[_owner].price==0){
        return false;
        }
        return true;
    }

    function getMyTicketsForEvent(string memory name) public view returns(uint){
        require(eventNames[name]==1,"No such event exists.");
        return events[eventOwners[name]].ticketHolders[msg.sender].total_tickets;
    }

    function createEvent(string memory name, uint total_tickets, uint price)
public{
        require(!checkForEventDeclaration(msg.sender),"You can only declare one
event at a time.");
        require(eventNames[name]==0,"Please select a different name for your
event.");
        events[msg.sender].name = name;
        events[msg.sender].total_tickets =
        total_tickets;
        events[msg.sender].price = price;
        eventNames[name] = 1;
        eventOwners[name] = msg.sender;
    }

    function closeEvent(string memory name)public{
        require(eventNames[name]==1,"Event doesn't exist."); address _owner =
eventOwners[name];
        require(_owner==msg.sender,"You do not have the priviledges to close this
event.");

        for(uint i=0;i<events[_owner].keys.length;i++){
            if(events[_owner].ticketHolders[events[_owner].keys[i]].total_tickets
>0){
                finalAttendees.push(events[_owner].keys[i]);
            }
            delete events[_owner].ticketHolders[events[_owner].keys[i]];

        }
        delete events[_owner];
        delete eventNames[name];
        delete eventOwners[name];
        delete finalAttendees;
```

```
    }

    function getTicketPrice(string memory name) public view returns(uint){
        require(events[eventOwners[name]].price>0,"No such eventexists.");
        return events[eventOwners[name]].price;
    }
    function buyFromEventOwner(string memory name, uint amt) payable public{
        address eventOwner = eventOwners[name];
        require(events[eventOwner].total_tickets>0,"Please try to buy from a peer
for this event.");
        require(events[eventOwner].total_tickets>=amt,stringAppend(uintToString(e
vents[eventOwner].total_tickets)," tickets left with the owner.Please request for
lesser number of tickets."));
        require(amt*events[eventOwner].price == msg.value,"Please provide proper
amount of Wei according to the price and the number of tickets to buy. Refer
getTicketPrice to knoe the price of tickets for your event.");
        events[eventOwners[name]].ticketHolders[msg.sender].total_tickets +=amt;
        events[eventOwners[name]].total_tickets -= amt;
        events[eventOwners[name]].keys.push(msg.sender);
        (bool success, ) = eventOwner.call{value:msg.value}("");
        require(success, "Error while transacting. Please try later");

    }

    function setForSale(string memory name, uint amt)public{
        require(eventNames[name]==1,"No such event exists.");
        address eventOwner = eventOwners[name];
        uint tmp = 0;

        for(uint i=0;i<events[eventOwner].keys.length;i++){
            if(events[eventOwner].keys[i] == msg.sender){
                tmp=1;
                break;
            }
        }

        require(tmp==1,"You do not have any tickets for this event.");
        require(events[eventOwner].ticketHolders[msg.sender].total_tickets>=amt,s
tringAppend("You
have",uintToString(events[eventOwner].ticketHolders[msg.sender].total_tickets)));
        events[eventOwner].ticketHolders[msg.sender].toSell = amt;
    }

    function removeForSale(string memory name) public{
```

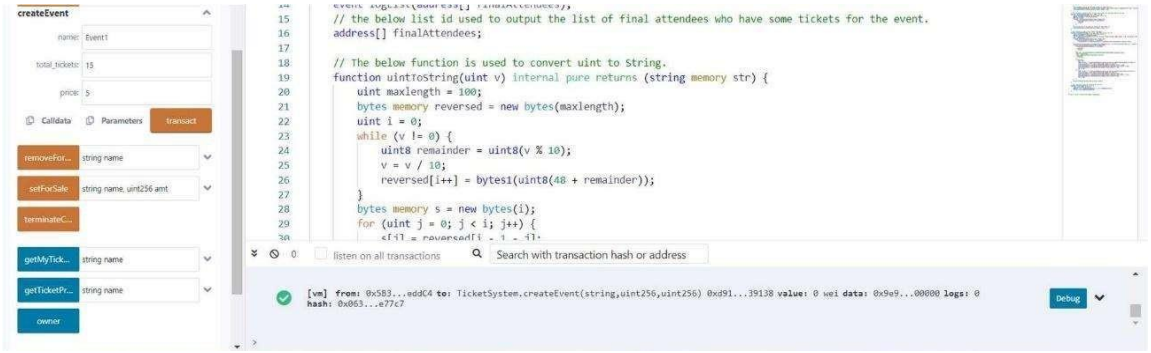
```
        require(eventNames[name]==1,"No such event exists."); address eventOwner
= eventOwners[name];
        bool tmp = false;
        for(uint i=0;i<events[eventOwner].keys.length;i++){
            if(events[eventOwner].keys[i] == msg.sender){
                tmp=true;
                break;
            }
        }
        require(tmp,"You do not have any tickets for this event.");
        events[eventOwner].ticketHolders[msg.sender].toSell = 0;
    }
    function buyFromPeer(string memory name, uint amt) payable public{
        require(eventNames[name]==1,"No such event exists.");
        address eventOwner = eventOwners[name];
        require(amt*events[eventOwner].price == msg.value,"Please provide proper
amount of Wei according to the price and the number of tickets to buy. Refer
getTicketPrice to knoe the price of tickets for your event.");
        uint totalWithPeers = 0;
        for(uint i=0;i<events[eventOwner].keys.length;i++){
            if(events[eventOwner].keys[i] != msg.sender)
            totalWithPeers
+=events[eventOwner].ticketHolders[events[eventOwner].keys[i]].toSell;
        }
        require(amt<=totalWithPeers,stringAppend(stringAppend("Only",uintToString
(totalWithPeers))," tickets left with peers."));
        for(uint i=0;i<events[eventOwner].keys.length;i++){
            if(events[eventOwner].keys[i] == msg.sender)
            continue;
            if(amt==0){
                break;
            }
            uint tmp =
events[eventOwner].ticketHolders[events[eventOwner].keys[i]].toSell;

            if(tmp==0){
                continue;
            }
            if(amt<tmp){
                tmp  =  amt;
                (bool success, )
=events[eventOwner].keys[i].call{value:tmp*events[eventOwner].price}("");
                require(success, "Error while transacting. Please trylater");
                events[eventOwner].ticketHolders[events[eventOwner].keys[i]].toSe
ll -= tmp;
```

```
                events[eventOwner].ticketHolders[events[eventOwner].keys[i]].tota
l_tickets -= tmp;
                events[eventOwner].ticketHolders[msg.sender].total_tickets +=tmp;
amt -= tmp;
            }
            else{
                (bool success, ) =
events[eventOwner].keys[i].call{value:tmp*events[eventOwner].price}("");
                require(success, "Error while transacting. Please try later");
                events[eventOwner].ticketHolders[events[eventOwner].keys[i]].toSe
ll = 0;
                events[eventOwner].ticketHolders[events[eventOwner].keys[i]].tota
l_tickets -= tmp;
                events[eventOwner].ticketHolders[msg.sender].total_tickets +=tmp;
amt -= tmp;
            }
        }
        events[eventOwners[name]].keys.push(msg.sender);
    }
    function terminateContract() public {
        require(msg.sender == owner);
        address payable sendRemainingEtherTo = payable(address(owner));
        selfdestruct(sendRemainingEtherTo);
    }
}
```

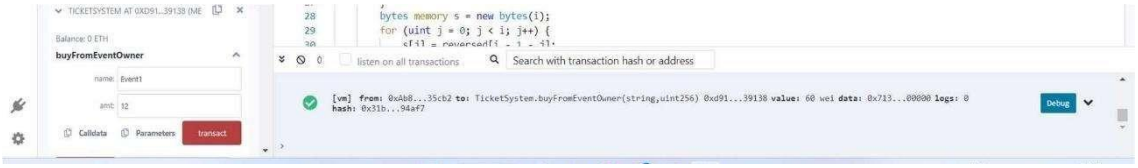
Output:

1) **Creating an Event.**



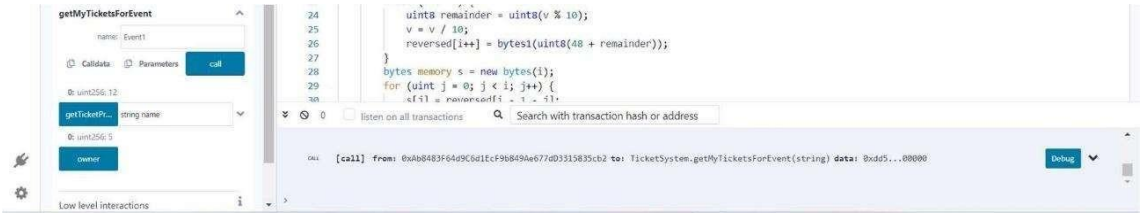
2) **Checking the price of the ticket for the created event.**

3) **Buying from Event Owner.**



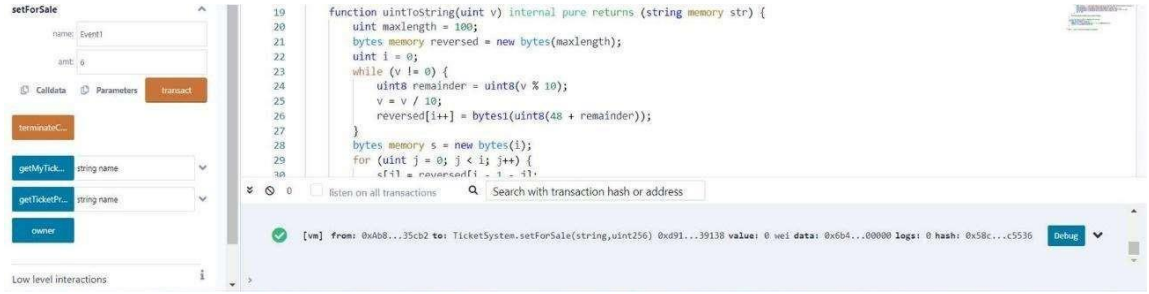
Here we need to be sure that we pass 60 Wei so as to buy 12 tickets as the price of each ticket is 5 Wei.

4) **Checking the number of tickets with us.**

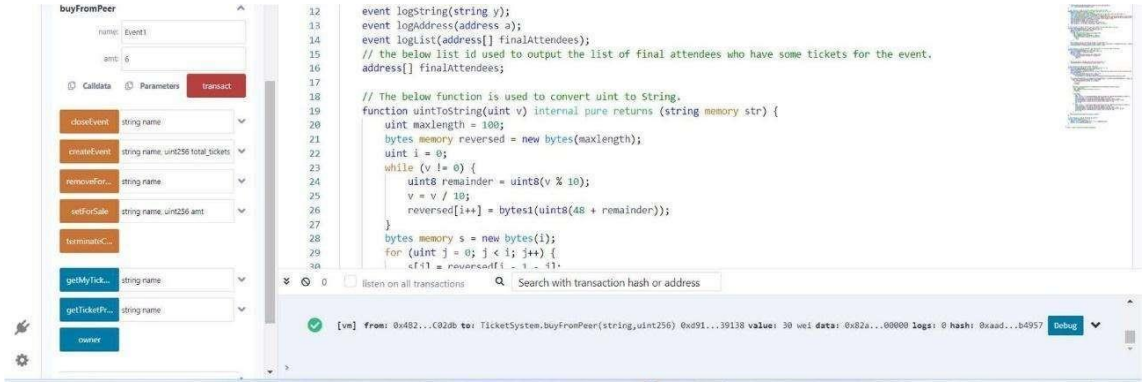


5) **Now we try to buy from the current user by switching to some other user via the buyFromPeer functionality.**

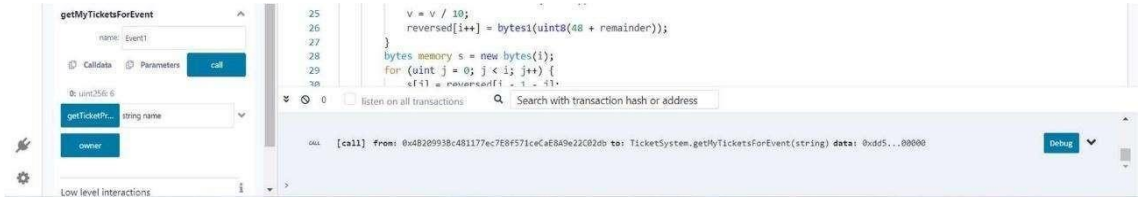
Firstly, we need to set toSell of the current user to lets assume 6 tickets, meaning that the current user is trying to sell 6 tickets to its peers.



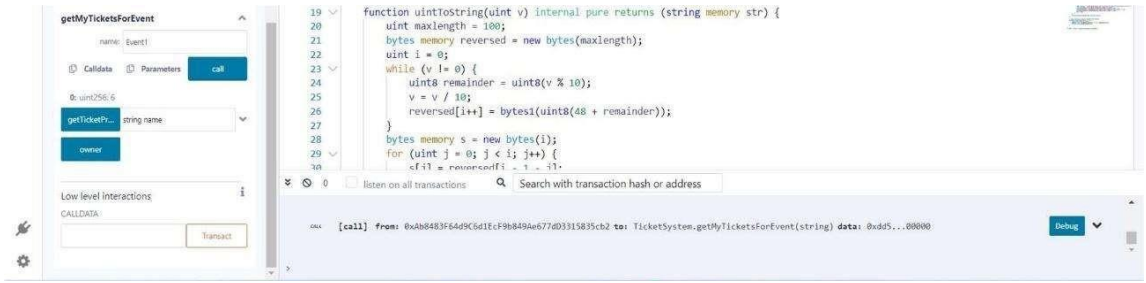
Now, switching the user and trying to buy from the peer.



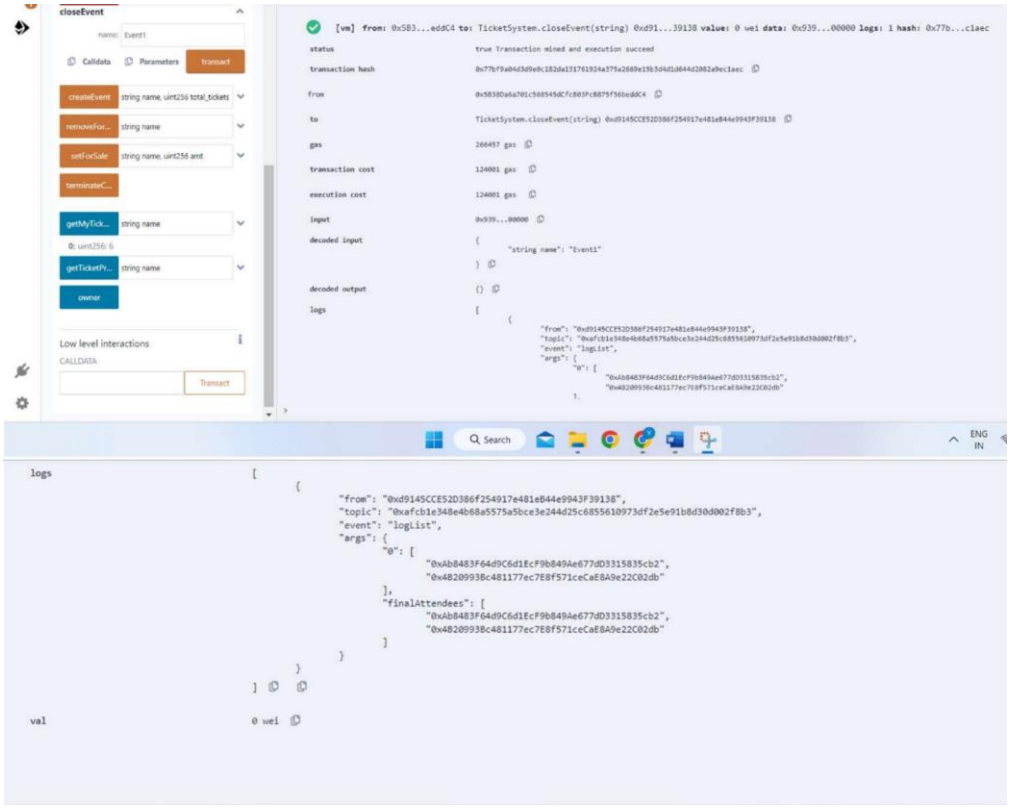
Now checking if the transaction is successful.



Also checking the deduction from the previous user.



6) **Lastly let's close the event and display the final list of users who own a ticket.**



7) Finally terminating the contract.