**Blockchain Technology**

**D2**

**19BCE248**

**Practical 5**

AIM: Implementing Smart Contract for your respective Term Paper topic

Topic: Smart Parking System

Code:

pragma solidity ^0.5.0;

contract SmartParking{

    struct Parking{

        bool isParked;

        uint price;

        bool isSpecial;

    }

    struct Person{

        bool isGuest;

        bool isAuthenticated;

    }

    Parking[] public parking;

    Person[] public person;

    function addPerson(bool isGuest,bool isAuthenticated) public {

        person.push(Person(isGuest,isAuthenticated));

    }

    function addParking(bool isParked,uint price,bool isSpecial) public {

        parking.push(Parking(isParked,price,isSpecial));

    }

    function getVacantPlace(bool isGuest) public view returns(uint){

            for(uint i=0;i<parking.length;i++){

                    if(!parking[i].isParked ){

                        if(isGuest && !parking[i].isSpecial){

                            return i;

                        }else if(!isGuest && parking[i].isSpecial){

                            return i;

                        }

                    }

            }

            return 1000;

    }

    function allocateParking(bool isGuest, bool isAuthenticated) public returns(bool){

        require(isAuthenticated);

        uint slotIndex=getVacantPlace(isGuest);

        if(slotIndex==1000){

            return false;

        }

        parking[slotIndex].isParked=true;

    }

}

Here we have implemented a very basic level of smart contract which covers most of the concept of solidity and to get familiar with the syntax as well as the remix environment.

There are mainly two stakeholders:

1. Person (Consumer)

* isGuest : Tells information about whether the particular person is daily visitor or guest.
* isAuthenticated: A very essential part to see whether he/she has a valid existential or are bind with some inappropriate activities which could further lead some problem to owner.

1. Parking (Supplier)

* isParked: Occupied/Vacant
* price: amount of parking slot
* isSpecial: Already reserved for daily customer or for the owners or pre-booked.

Similary we have three functions namely:

1. addPerson : Called whenever a new person arrives at entry gate
2. addParking : Pre-defined at the time of construction
3. getVacantPlace : According to current snapshot return index of vacant places
4. allocateParking : getVacantPlace check for authenticity and allocate parking if all the requirements fulfilled.

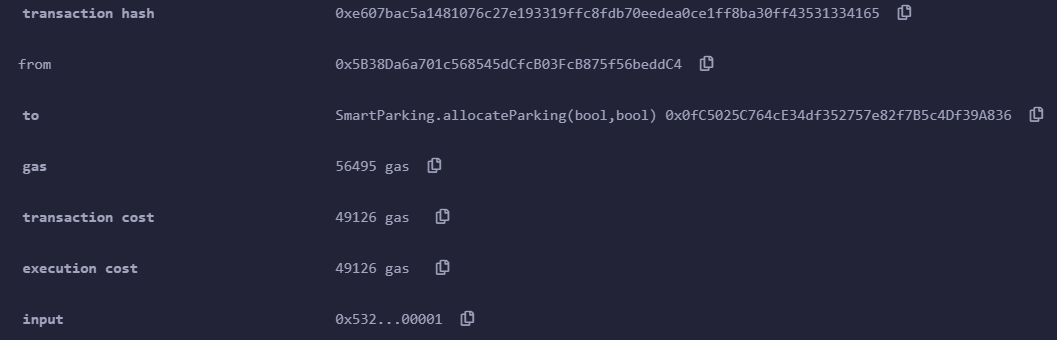
Output:

Parking:

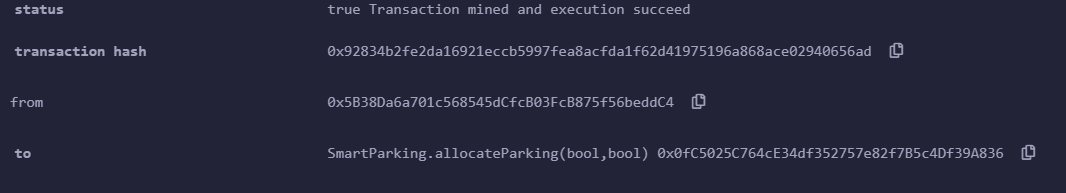
1. false,35,false
2. false,35,false
3. false,35,false
4. false,50,true
5. false,50,true

allocateParking:

1. true,true



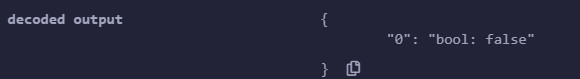
1. true,true



1. true,true

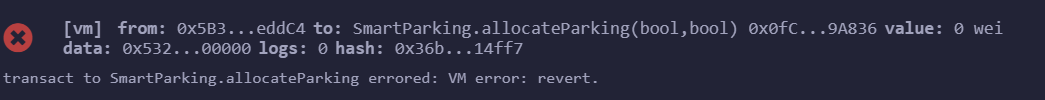
Will again allocate

1. true,true



Deny

1. false,false



Error due to not authenticate person