**Project: Cab Ride**

As of now, when riders book the cab, the notification is first sent to the company and then the driver is allocated for a specific ride. The cab companies charge 10-20 percent of the total ride’s pricing, costing high to the customers. Therefore, the involvement of intermediaries between a driver and a rider may result in the additional costs.

Blockchain would enable riders to directly connect with drivers through the decentralized platform, reducing the additional costs incurred due to the involvement of too many intermediaries.

# Solidity Codes:

## Token creation:

pragma solidity 0.6.12;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol";

contract RideToken is ERC20 {

address public owner;

constructor(string memory name, string memory symbol, uint256 count) ERC20(name, symbol) public {

owner = msg.sender;

\_mint(msg.sender, count \* 10 \*\* 18);

}

}

## Code which performs various functions involved in Cab ride

pragma solidity 0.6.12;

//import "./RideICO.sol";

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol";

contract testRide {

struct request {

string name;

string pick\_loc;

string drop\_loc;

bool accepted;

uint tokenNum;

uint256 create\_time;

address cust\_address;

uint256 requestid;

}

struct customer {

string cname;

}

struct driver {

string name;

int256 rating;

string number\_plate;

address driveraddress;

}

mapping (address => customer) customers;

mapping (address => driver) cab\_drivers;

mapping (address => request) requests;

address[] driver\_addresses;

address[] customer\_addresses;

address[] request\_addresses;

event RideGiven(uint256 rideId, address driver\_add);

event RideStatus(uint256 rideId, address driver\_add,string currentstatus);

customer c1;

driver d1;

address contractaddress = 0x583031D1113aD414F02576BD6afaBfb302140225; // contract address

bool finaltranfer;

//Register Drivers

function RegisterDriver(string memory name, string memory number\_plate,address driveraddress) public

{

d1.name = name;

d1.number\_plate = number\_plate;

//d1.driveraddress = driveraddress;

cab\_drivers[msg.sender] = d1;

driver\_addresses.push(msg.sender);

}

//Register Customer

function RegisterCustomer(string memory cname) public

{

c1.cname = cname;

customers[msg.sender] = c1;

}

// function checkExistenceDriver() private returns(string memory)

// {

// if(keccak256(cab\_drivers[msg.sender].name) == keccak256(""))

// return "false";

// return "true";

// }

uint tokenNum;

address driveradd;

// Request Ride

request r1;

function RequestRide(string memory name,string memory pickup, string memory drop) public returns (string memory , uint256 )

{

//check for registered customers

require(keccak256(abi.encodePacked(customers[msg.sender].cname)) == keccak256(abi.encodePacked(name)),"Cutomer is not registered");

r1.name=name;

r1.pick\_loc = pickup;

r1.drop\_loc = drop;

r1.accepted = false;

r1.cust\_address = msg.sender;

r1.create\_time = now;

r1.requestid += 1;

requests[msg.sender] = r1;

request\_addresses.push(msg.sender);

uint256 temprequest = r1.requestid;

emit RideGiven(r1.requestid, msg.sender);

return("Your Ride Id is:",temprequest);

}

// Start the Ride

function startRide(uint \_rideNumber,bool driver\_accept,uint price,address driveraddress) public {

// check for the confirmation from driver

//transfer the token to driver once the ride is completed

require( r1.requestid == \_rideNumber, "Wrong Request Id entered,please try again with correct request Id!!");

require( driver\_accept == true, "Ride Not confirmed!!");

tokenNum = price;

driveradd = driveraddress;

emit RideStatus(r1.requestid, msg.sender,"Ride Confirmed");

TempTransferTokens(r1.cust\_address,contractaddress,tokenNum);

}

// End the Ride

function endRide(uint \_rideNum,bool passenger\_accept,uint Tip) public {

require( r1.requestid == \_rideNum, "Wrong Request Id entered,please try again with correct request Id!!");

//require( r1.accepted == true, "Ride Not started yet!!");

require( passenger\_accept == true, "Ride completed!!");

tokenNum = tokenNum + Tip;

// call transfretoken contract

finaltranfer= true;

r1.accepted = true;

TransferTokens(contractaddress,driveradd,tokenNum);

emit RideStatus(r1.requestid, msg.sender,"Ride Completed");

delete r1;

}

function cancelRide(uint rideNum) public {

require( r1.requestid == rideNum, "Wrong Request Id entered,please try again with correct request Id!!");

require( r1.accepted == false, "Ride already confirmed and can not be Cancelled!!");

emit RideStatus(r1.requestid, msg.sender,"Ride Cancelled");

//withdraw amount

TransferTokens(contractaddress,r1.cust\_address,tokenNum);

delete r1;

}

// buy tokens funtionality

uint tokensPerEther = 100;

uint weiToEther = 1000000000000000000;

IERC20 token;

event TokensBought(address owner,address who, uint num);

event TokensTransfer(address who, uint num);

address owner;

constructor() public {

owner = msg.sender;

}

function setToken(address \_token) public {

token = IERC20(\_token);

}

function buyTokens(uint256 count) public payable {

uint actualTokens = msg.value / weiToEther\* tokensPerEther;

require(count <= actualTokens, "You've sent less money than required for buying specified tokens");

token.transferFrom(owner,msg.sender,count);

emit TokensBought(owner,msg.sender, count);

}

function TransferTokens(address \_cust\_address,address \_driveradd, uint \_cnt) internal {

// address driveradd = 0xdD870fA1b7C4700F2BD7f44238821C26f7392148 ;

token.transferFrom(\_cust\_address,\_driveradd,\_cnt);

emit TokensBought(\_cust\_address,\_driveradd,\_cnt);

}

//temp TransferTokens to contract address

function TempTransferTokens(address \_fromadr,address \_toadr, uint tcount) internal {

// address driveradd = 0xdD870fA1b7C4700F2BD7f44238821C26f7392148 ;

token.transferFrom(\_fromadr,\_toadr,tcount);

emit TokensBought(\_fromadr,\_toadr,tcount);

finaltranfer=false;

}

}