EV Charging Station

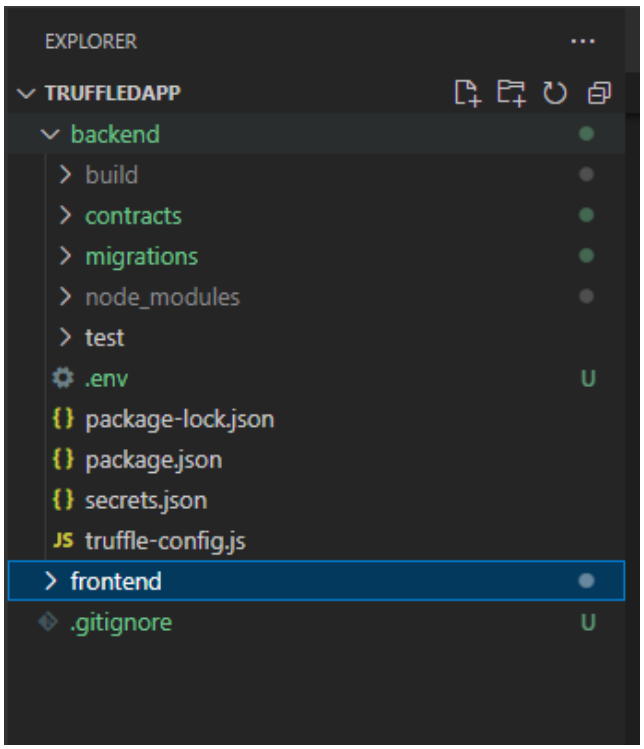
IT832: Blockchain Technologies and Applications- Decentralization and Smart Contracts

Assignment I

222IT008-Deepak Gaur

222IT007-Tushar Chaudhari

Backend developed using **Solidity** programming language.
The code base structure looks like :



The entire project is uploaded on Github.
Repository link : https://github.com/tushar-chaudhari/Blockchain-Assignment-1

Steps to deploy smart-contract:

1. Configure truffle-config.js file according to Network
2. Configure .env file to store **Metamask secret key** along with **API** key of third party which helps for deployment of Blockchain application.
3. Go to Backend directory and open terminal/cmd.
4. Type **"truffle compile"** and execute the command
5. Type "truffle migrate –network goerli" and note the contract address.

6. The contract address and ABI can be found **build/contracts/contract_name.json** file
7. Use this contract address and ABI in frontend to connect with backend.

Some Important files are :

1. truffle-config.js
2. EVContract.sol
3. 1_deploy_evcontract.js
4. .env

⇒ truffle-config.js

```
require('dotenv').config();
const HDWalletProvider = require('@truffle/hdwallet-provider');
const { INFURA_API_KEY, MNEMONIC } = process.env;

module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*"
    },
    goerli: {
      provider: () => new HDWalletProvider(MNEMONIC, INFURA_API_KEY),
      network_id: '5',
      gas: 4465030
    }
  }
};
```

⇒ EVContract.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.22 <0.9.0;
pragma experimental ABIEncoderV2;

contract EVContract {
  uint public count = 0;
  uint[] allottedStationsID;
  charging_station[] charging_stations_array;
```

```
  EVCarOwner[] private evOwnerInfo;

  struct EVCarOwner {
    uint id;
    address username;
    uint range;
    string preference;
    bool fastChargingNeeded;
    bool otherBrandChargingStation;
    uint[] allotedChargingStations;
  }
    struct charging_station {
      uint station_id; //0
      string station_name; //0
      uint remaining_distance;
      uint price_per_unit;
      uint count_of_waiting_cars;
      bool fast_charging;
      bool other_company_station;
      uint rating;
    }

  mapping(uint256 => address) public evOwnerRequests;
  event SuggestChargingStation(address recipient,uint taskID);

  constructor() public{
    addChargingStation(1,'Kunjathbail Road',12,12,11,true,false,1);
    addChargingStation(2,'Lalbag Kodical',27,11,2,false,true,2);
    addChargingStation(3,'Akasha Bhavana
Chowk',29,19,15,true,false,5); //7
    addChargingStation(4,'Anandanagar Main
Road',25,11,4,true,true,4); //5
    addChargingStation(5,'Shaktinagar Near
Hospital',18,15,9,false,true,2); //2

    addChargingStation(6,'KUNJATHBAIL',14,11,2,false,true,1); //1
    addChargingStation(7,'Neermarga',40,10,7,false,true,5); //2
    addChargingStation(8,'Jalligudde Tata',25,18,11,true,true,1);
//7
    addChargingStation(9,'Akasha Bhavana',26,11,0,false,true,3); //5
```

```
    addChargingStation(10,'Dambel HP',7,16,8,false,false,2); //2

    addChargingStation(11,'Neermarga Power
Grid',42,11,6,true,false,1); //1
    addChargingStation(12,'Anandanagar',24,19,6,true,true,5); //2
    addChargingStation(13,'Surathkal
Kaikamba',21,20,4,true,false,5); //7
    addChargingStation(14,'Moodshedde',37,16,13,true,false,4); //5
    addChargingStation(15,'Surathkal',15,13,0,false,false,5); //2

    addChargingStation(16,'Jalligudde',24,20,15,false,true,5); //1
    addChargingStation(17,'Dambel Tata',8,15,13,true,true,1); //2
    addChargingStation(18,'Shaktinagar',19,10,3,false,false,3); //7
    addChargingStation(19,'Shaktinagar MG',20,19,11,true,true,4);
//5
    addChargingStation(20,'Kunjathbail',70,14,6,true,true,3); //2

  }


  function suggestChargingStation(uint _range,string memory
_preference, bool _fastChargingNeeded,bool
_other_brand_charging_station) public {
   uint taskId = evOwnerInfo.length;
    evOwnerInfo.push(EVCarOwner(taskId, msg.sender,
_range,_preference,_fastChargingNeeded,
_other_brand_charging_station,getAllotedStations(_range,_preference,
_fastChargingNeeded,_other_brand_charging_station)));
    evOwnerRequests[taskId] = msg.sender;
    emit SuggestChargingStation(msg.sender, taskId);
  }

  function getAllotedStations(uint _range,string memory _preference,
bool _fastChargingNeeded,bool _other_brand_charging_station)
internal returns (uint[] memory) {
      delete allottedStationsID;
      // need to write real logic here
      charging_station[] memory temp;
```

```
        charging_station[] memory cs =
getFilteredChargingStations(_range,_fastChargingNeeded,_other_brand_
charging_station);
        if(keccak256(bytes(_preference)) ==
keccak256(bytes("distance"))){
            temp = sortByRemaingDistance(cs);
        }else{
            temp = sortByPrice(cs);
        }
        for(uint i=0; i < temp.length; i++){
            allottedStationsID.push(temp[i].station_id);
        }
      return allottedStationsID;
  }

  function getCurrentRequestData() public view returns(EVCarOwner
memory){
      uint maxLenght = evOwnerInfo.length;
      return evOwnerInfo[maxLenght - 1];
  }

  function getUserRequests() external view returns (EVCarOwner[]
memory){
    EVCarOwner[] memory temporary = new
EVCarOwner[](evOwnerInfo.length);
    uint counter = 0;
    for(uint i=0; i<evOwnerInfo.length; i++) {
        if(evOwnerRequests[i] == msg.sender &&
evOwnerInfo[i].allotedChargingStations.length != 0) {
            temporary[counter] = evOwnerInfo[i];
            counter++;
          }
        }
      EVCarOwner[] memory result = new EVCarOwner[](counter);
      for(uint i=0; i<counter; i++) {
          result[i] = temporary[i];
      }
    return result;
  }
```

```
  function getInfoCS() public view returns (charging_station[]
memory){
      return charging_stations_array;
  }

    function addChargingStation(uint station_id,string memory
station_name,uint remaining_distance,uint price_per_unit,uint
count_of_waiting_cars,bool fast_charging,bool
other_charging_station,uint rating) public {
        count++;

charging_stations_array.push(charging_station(station_id,station_nam
e,remaining_distance,price_per_unit,count_of_waiting_cars,fast_charg
ing,other_charging_station,rating));
    }

    function getChargingStationDetails(uint _id) public view
returns(charging_station memory){
        uint i = 0;
        for (; i<=charging_stations_array.length; i++) {
            if(charging_stations_array[i].station_id == _id){
                break;
            }
        }
        return charging_stations_array[i];
    }

    function getFilteredChargingStations(uint _range,bool
_fast_charging_support,bool _other_brand_charging_station) public
view returns(charging_station[] memory){
        charging_station[] memory cs = getInfoCS();
        charging_station[] memory temporary = new
charging_station[](cs.length);
        uint counter = 0;

        for(uint i = 0; i < cs.length; i++) {
            if(cs[i].remaining_distance <= _range &&
cs[i].fast_charging == _fast_charging_support &&
cs[i].other_company_station == _other_brand_charging_station) {
                temporary[counter] = cs[i];
```

```
                counter++;
            }
        }
        charging_station[] memory result = new
charging_station[](counter);
        for(uint i=0; i < counter; i++){
            result[i] = temporary[i];
        }
        return result;
    }

   function sortByRemaingDistance(charging_station[] memory cs)
public pure returns(charging_station[] memory) {
        //charging_station[] memory cs = getInfoCS();
        for (uint i = 1; i < cs.length; i++)
            for (uint j = 0; j < i; j++)
                if (cs[i].remaining_distance <
cs[j].remaining_distance) {
                    charging_station memory x = cs[i];
                    cs[i] = cs[j];
                    cs[j] = x;
                }else if(cs[i].remaining_distance ==
cs[j].remaining_distance){
                    if(cs[i].price_per_unit < cs[j].price_per_unit){
                        charging_station memory x = cs[j];
                        cs[j] = cs[i];
                        cs[i] = x;
                    }
                    else if(cs[i].price_per_unit ==
cs[j].price_per_unit){
                         if(cs[i].rating > cs[j].rating){
                            charging_station memory x = cs[j];
                            cs[j] = cs[i];
                            cs[i] = x;
                        }else if(cs[i].rating == cs[j].rating){
                            if(cs[i].count_of_waiting_cars <
cs[j].count_of_waiting_cars){
                                charging_station memory x = cs[j];
                                cs[j] = cs[i];
                                cs[i] = x;
```

```
                            }
                        }
                    }
                }
        return cs;
    }

   function sortByPrice(charging_station[] memory cs) public pure
returns(charging_station[] memory) {
        for (uint i = 1; i < cs.length; i++)
            for (uint j = 0; j < i; j++)
                 if(cs[i].price_per_unit < cs[j].price_per_unit){
                        charging_station memory x = cs[j];
                        cs[j] = cs[i];
                        cs[i] = x;
                }else if(cs[i].price_per_unit ==
cs[j].price_per_unit){
                        if (cs[i].remaining_distance <
cs[j].remaining_distance) {
                            charging_station memory x = cs[i];
                            cs[i] = cs[j];
                            cs[j] = x;
                        }
                        else if(cs[i].remaining_distance ==
cs[j].remaining_distance){
                            if(cs[i].rating > cs[j].rating){
                                charging_station memory x = cs[j];
                                cs[j] = cs[i];
                                cs[i] = x;
                            }else if(cs[i].rating == cs[j].rating){
                                if(cs[i].count_of_waiting_cars <
cs[j].count_of_waiting_cars){
                                        charging_station memory x =
cs[j];
                                        cs[j] = cs[i];
                                        cs[i] = x;
                                }
                            }

                        }
```

```
                }

        return cs;
    }
}
```

⇒ 1_deploy_evcontract.js

```
const Demo_Contract = artifacts.require("./EVContract.sol");

module.exports = function(deployer) {
  deployer.deploy(Demo_Contract);
};
```

⇒ .env

It contains private keys.