//SPDX-License-Identifier:MIT

pragma solidity >=0.7.0 <0.9.0;

contract DoctorToken {

//address of the owner of the smart contract

address private owner;

//Auto generated doctorID ID

uint private doctorIDNo;

//Variable to keep track of TotalNoOfAgencies

uint private TotalNoOfAgencies;

//map a number to an Agency address to Agency

mapping(address=>HealthAgency) private agencyCollection;

DoctorDetails[] public doctorCollection;

//Array list for Patients

PatientData[] private patientData;

//Enumerated list for status of Doctor

enum DoctorStatus{Created,Verified,Delivered}

enum SuccessRate{Total,Partial,UnSatisfied}

int[3] tokenValue=[-5,5,10];

//event DoctorVerifiedEvent(address,uint);

event DoctorVerifiedEvent(address,uint);

//Event for emitting Doctor ID

event DoctorID(uint);

//Structure for doctor

struct DoctorDetails{

int TokenValue;

bytes32 Remarks;

bytes32 DoctorName;

uint DoctorAge;

bytes32 DoctorCaseDetails;

uint DoctorContactNo;

//Id used for medical use , will be auto generated

uint DoctorId;

//{Created,Verified,Delivered}

DoctorStatus Status;

//date , it will auto set

bytes32 DateCreated;

//it will be updated when ever we will update something about the Doctor

bytes32 DateUpdated;

//agency associate this blood

address agency;

}

//structure to represent Patient Data

struct PatientData{

bytes32 Remarks;

bytes32 Name;

uint Age;

bytes32 MedicalDetails;

uint ContactNo;

bytes32 CaseDetails;

bytes32 DateProvided;

address agency;

// string agencyType;

string doctorDetails;

}

//Agency strcture to store agency information , agency could be hospital, Government Agency

struct HealthAgency {

string agencyType;

string agencyName;

address addressID;

string contactPerson;

uint contactNo;

string agencyAddress;

uint pincode;

}

//constructor called ones at the time of deploy

constructor() {

//storing the contract deployer address as owner

owner = msg.sender;

}

/\*\* Returns the current total agency available. \*/

function Agency\_Total() public view returns (uint) {

return TotalNoOfAgencies;

}

function addDoctorDetails(

bytes32 \_Remarks,

bytes32 \_DoctorName,

uint \_DoctorAge,

bytes32 \_DoctorCaseDetails,

uint \_DoctorContactNo,

bytes32 \_DateCreated,

bytes32 \_DateUpdated)

public {

require(agencyCollection[msg.sender].addressID != 0x0000000000000000000000000000000000000000,"Sorry you are not authorized");

doctorCollection.push(DoctorDetails(0,\_Remarks,\_DoctorName,\_DoctorAge,\_DoctorCaseDetails,

\_DoctorContactNo,++doctorIDNo,DoctorStatus.Created,

\_DateCreated,\_DateUpdated,msg.sender));

emit DoctorID(doctorIDNo);

}

function getDoctorList(uint start) public view returns(DoctorDetails[10] memory temp){

for(uint i = 0; i < start + 10 && start + i < doctorCollection.length; i++){

temp[i] = doctorCollection[start + i];

}

return temp;

}

function getDoctorTokenCount(uint i) public view returns(int temp){

temp = doctorCollection[i].TokenValue;

return temp;

}

function verifyDoctorDetails(uint \_DoctorID) public{

require(owner==msg.sender,"You are not authorized to Verify");

require(\_DoctorID != 0 , "No Doctor With This Id");

//as we are accessing by index

\_DoctorID-=1;

require(doctorCollection[\_DoctorID].Status == DoctorStatus.Created, "Not created yet");

doctorCollection[\_DoctorID].Status = DoctorStatus.Verified;

emit DoctorVerifiedEvent(msg.sender,\_DoctorID+1);

}

function AddTokenToDoctor(

uint \_DoctorId, bytes32 \_Remarks,bytes32 \_Name,uint \_Age,bytes32 \_MedicalDetails,uint \_ContactNo,

string memory \_doctorDetails,bytes32 \_DateDemanded,bytes32 \_DateProvided,SuccessRate \_SuccRate

)

public

{

//First check for owner

require(owner==msg.sender,"You are not authorized to Verify");

//Check that BottleiD demanded is not Zero

require(\_DoctorId != 0 , "No Doctor With This Id");

//to access it by indexed

\_DoctorId-=1;

//Check for status of BottleID Verified or not

require(doctorCollection[\_DoctorId].Status == DoctorStatus.Verified, "Not Verified yet");

patientData.push(PatientData(\_Remarks,\_Name,\_Age,\_MedicalDetails,\_ContactNo,\_DateDemanded,\_DateProvided,msg.sender,\_doctorDetails));

if(\_SuccRate == SuccessRate.Total)

{

doctorCollection[\_DoctorId].TokenValue +=tokenValue[2];

}

else

if(\_SuccRate == SuccessRate.Partial)

{

doctorCollection[\_DoctorId].TokenValue +=tokenValue[1];

}

else

if(\_SuccRate == SuccessRate.UnSatisfied)

{

doctorCollection[\_DoctorId].TokenValue +=tokenValue[0];

}

}

//add agency to the agency collection

function addHealthAgency(string memory \_agencyType,string memory \_agencyName,

address \_addressID,string memory \_contactPerson,

uint \_contactNo,string memory \_agencyAddress,uint \_pincode) public {

require(msg.sender==owner,"Sorry you are not authorized");

require(agencyCollection[\_addressID].addressID == 0x0000000000000000000000000000000000000000, "Duplicate Agency Id");

// create an temporary agency object

HealthAgency memory agency;

agency.agencyType=\_agencyType;

agency.agencyName = \_agencyName;

agency.addressID = \_addressID;

agency.contactPerson = \_contactPerson;

agency.contactNo = \_contactNo;

agency.agencyAddress = \_agencyAddress;

agency.pincode =\_pincode;

agencyCollection[\_addressID] = agency;

TotalNoOfAgencies++;

}

//get agency data

function getHealthAgencyData(address \_addressID) public view returns(HealthAgency memory agency){

agency = agencyCollection[\_addressID];

return agency;

}

}