pragma solidity ^0.5.1;

pragma experimental ABIEncoderV2;

contract Hospital\_Trainee\_Competency {

address public \_NationalAuthority;

struct Hospital {

address \_HospitalAddress;

uint \_HospitalID;

string \_HospitalName;

}

Hospital H;

mapping (address => Hospital) Hos;

Hospital[] ListOfHospital;

struct Trainee {

address \_TraineeAddress;

uint \_TraineeID;

string \_TraineeName;

string \_TrHospitaName;

string \_Category;

string \_TraineeCertificate;

}

Trainee T;

mapping (address => Trainee) Tra;

mapping (address => Trainee) Tr;

Trainee[] ListOfTrainee;

Trainee[] TraineeRecords;

address[] public OfficialEmployees;

event delete\_Hospital(address \_HospitalAddress,uint \_HospitalID);

event add\_Records(address \_HospitalAddress, address \_TraineeAddress, uint \_TraineeID, string \_TraineeName, string \_TrHospitaName, string \_Category, string \_TraineeCertificate);

constructor () public {

\_NationalAuthority = msg.sender;

}

function Authorise\_Hospital(address \_HospitalAddress, uint \_HospitalID, string memory \_HospitalName) public {

require(msg.sender == \_NationalAuthority);

Hospital storage hos = Hos[\_HospitalAddress];

require(\_HospitalAddress != hos.\_HospitalAddress);

require(\_HospitalID != hos.\_HospitalID);

hos.\_HospitalAddress = \_HospitalAddress;

hos.\_HospitalID = \_HospitalID;

hos.\_HospitalName = \_HospitalName;

ListOfHospital.push(hos);

}

function Add\_Employees(address \_HospitalAddress, address \_TraineeAddress, uint \_TraineeID, string memory \_TraineeName, string memory \_TrHospitalName) public {

require(Hos[\_HospitalAddress].\_HospitalAddress == msg.sender);

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

require(OfficialEmployees[i] != \_TraineeAddress);

}

OfficialEmployees.push(\_TraineeAddress);

Trainee storage tra = Tra[\_TraineeAddress];

tra.\_TraineeAddress = \_TraineeAddress;

tra.\_TraineeID = \_TraineeID;

tra.\_TraineeName = \_TraineeName;

tra.\_TrHospitaName = \_TrHospitalName;

ListOfTrainee.push(tra);

}

function Add\_Records(address \_HospitalAddress, address \_TraineeAddress, uint \_TraineeID, string memory \_TraineeName, string memory \_TrHospitaName, string memory \_Category, string memory \_TraineeCertificate)public {

require(Hos[\_HospitalAddress].\_HospitalAddress == msg.sender);

require(keccak256(abi.encodePacked(Hos[\_HospitalAddress].\_HospitalName)) == keccak256(abi.encodePacked(Tra[\_TraineeAddress].\_TrHospitaName)));

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

{

if(OfficialEmployees[i] == \_TraineeAddress)

{

Trainee storage tra = Tra[\_TraineeAddress];

tra.\_TraineeAddress = \_TraineeAddress;

tra.\_TraineeID = \_TraineeID;

tra.\_TraineeName = \_TraineeName;

tra.\_TrHospitaName = \_TrHospitaName;

tra.\_Category = \_Category;

tra.\_TraineeCertificate = \_TraineeCertificate;

ListOfTrainee.push(tra);

emit add\_Records(\_HospitalAddress, \_TraineeAddress, \_TraineeID, \_TraineeName, \_TrHospitaName, \_Category, \_TraineeCertificate);

break;

}

}

}

function deleteHospital(address \_HospitalAddress, uint \_HospitalID) public {

require(msg.sender == \_NationalAuthority);

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

{

if(\_HospitalAddress == ListOfHospital[i].\_HospitalAddress && \_HospitalID == ListOfHospital[i].\_HospitalID)

{

delete ListOfHospital[i];

emit delete\_Hospital(\_HospitalAddress, \_HospitalID);

break;

}

}

}

function deleteEmployee(address \_HospitalAddress, address \_TraineeAddress, uint \_TraineeID) public {

require(msg.sender == Hos[\_HospitalAddress].\_HospitalAddress);

/\*

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

require(OfficialEmployees[i] != \_TraineeAddress);

}

\*/

require(keccak256(abi.encodePacked(Hos[\_HospitalAddress].\_HospitalName)) == keccak256(abi.encodePacked(Tra[\_TraineeAddress].\_TrHospitaName)));

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

if(ListOfTrainee[i].\_TraineeAddress == \_TraineeAddress && ListOfTrainee[i].\_TraineeID == \_TraineeID)

{

delete ListOfTrainee[i];

break;

}

}

}

function splitSignature(bytes memory sig)

internal pure

returns (uint8 v, bytes32 r, bytes32 s)

{

require(sig.length == 65);

assembly {

// first 32 bytes, after the length prefix.

r := mload(add(sig, 32))

// second 32 bytes.

s := mload(add(sig, 64))

// final byte (first byte of the next 32 bytes).

v := byte(0, mload(add(sig, 96)))

}

return (v, r, s);

}

function recoverSigner(bytes32 message, bytes memory sig)

internal

pure

returns (address)

{

(uint8 v, bytes32 r, bytes32 s) = splitSignature(sig);

return ecrecover(message, v, r, s);

}

function getListOfHospitals() public view returns(Hospital[] memory){

return ListOfHospital;

}

function See\_Authorised\_Hospital(address \_HospitalAddress) view public returns (address, uint, string memory){

return (Hos[\_HospitalAddress].\_HospitalAddress, Hos[\_HospitalAddress].\_HospitalID, Hos[\_HospitalAddress].\_HospitalName);

}

function See\_Employee\_Records(address \_TraineeAddress) view public returns (uint, string memory, string memory, string memory, string memory){

return (Tra[\_TraineeAddress].\_TraineeID, Tra[\_TraineeAddress].\_TraineeName, Tra[\_TraineeAddress].\_TrHospitaName, Tra[\_TraineeAddress].\_Category, Tra[\_TraineeAddress].\_TraineeCertificate);

}

function getListOfTraineesRecords(address \_TraineeAddress) public returns(Trainee[] memory) {

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

if(ListOfTrainee[i].\_TraineeAddress == \_TraineeAddress){

Trainee storage T = Tr[\_TraineeAddress];

T.\_TraineeAddress = ListOfTrainee[i].\_TraineeAddress;

T.\_TraineeID = ListOfTrainee[i].\_TraineeID;

T.\_TraineeName = ListOfTrainee[i].\_TraineeName;

T.\_TrHospitaName = ListOfTrainee[i].\_TrHospitaName;

T.\_Category = ListOfTrainee[i].\_Category;

T.\_TraineeCertificate = ListOfTrainee[i].\_TraineeCertificate;

TraineeRecords.push(T);

}

return TraineeRecords;

}

}

}