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;

string \_Signature;

}

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, string \_Signature);

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);

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

if(keccak256(abi.encodePacked(ListOfTrainee[i].\_TrHospitaName)) == keccak256(abi.encodePacked(Tra[\_TraineeAddress].\_TrHospitaName))) {

require(\_TraineeID != ListOfTrainee[i].\_TraineeID);

}

}

Trainee storage tra = Tra[\_TraineeAddress];

tra.\_TraineeAddress = \_TraineeAddress;

tra.\_TraineeID = \_TraineeID;

tra.\_TraineeName = \_TraineeName;

tra.\_TrHospitaName = \_TrHospitalName;

tra.\_Category = "None";

tra.\_TraineeCertificate = "None";

tra.\_Signature = "None";

ListOfTrainee.push(tra);

}

function Add\_Records(address \_HospitalAddress, address \_TraineeAddress, uint \_TraineeID, string memory \_TraineeName, string memory \_TrHospitaName, string memory \_Category, string memory \_TraineeCertificate, string memory \_Signature)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];

require(Tra[\_TraineeAddress].\_TraineeAddress == \_TraineeAddress);

require(Tra[\_TraineeAddress].\_TraineeID == \_TraineeID);

if((keccak256(abi.encodePacked(Tra[\_TraineeAddress].\_TraineeName)) == keccak256(abi.encodePacked(\_TraineeName)) && keccak256(abi.encodePacked(Tra[\_TraineeAddress].\_TrHospitaName)) == keccak256(abi.encodePacked(\_TrHospitaName)))){

tra.\_TraineeAddress = \_TraineeAddress;

tra.\_TraineeID = \_TraineeID;

tra.\_TraineeName = \_TraineeName;

tra.\_TrHospitaName = \_TrHospitaName;

tra.\_Category = \_Category;

tra.\_TraineeCertificate = \_TraineeCertificate;

tra.\_Signature = \_Signature;

ListOfTrainee.push(tra);

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

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++){

if(OfficialEmployees[i] == \_TraineeAddress){

delete OfficialEmployees[i];

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

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

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

{

delete ListOfTrainee[i];

break;

}

}

}

}

}

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

return ListOfHospital;

}

function getListOfTrainees() public view returns (Trainee[] memory){

return ListOfTrainee;

}

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

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

}

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

uint256 HospitalID = Hos[\_HospitalAddress].\_HospitalID;

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

}

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

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

}

function getListOfTraineesRecords(address \_TraineeAddress) public {

delete TraineeRecords;

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

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

Trainee storage Trai = Tr[\_TraineeAddress];

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

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

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

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

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

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

Trai.\_Signature = ListOfTrainee[i].\_Signature;

TraineeRecords.push(Trai);

}

}

}

function Records() public view returns(Trainee[] memory){

return TraineeRecords;

}

}

contract SignatureVerifier {

address public owner;

constructor() public {

owner = msg.sender;

}

function verifySignature(bytes32 hash, bytes memory signature, address signer) public pure returns (bool) {

address addressFromSig = recoverSigner(hash, signature);

return addressFromSig == signer;

}

function recoverSigner(bytes32 hash, bytes memory sig) public pure returns (address) {

require(sig.length == 65, "Require correct length");

bytes32 r;

bytes32 s;

uint8 v;

assembly {

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

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

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

}

// Version of signature should be 27 or 28, but 0 and 1 are also possible versions

if (v < 27) {

v += 27;

}

require(v == 27 || v == 28, "Signature version not match");

return recoverSigner2(hash, v, r, s);

}

function recoverSigner2(bytes32 h, uint8 v, bytes32 r, bytes32 s) public pure returns (address) {

bytes memory prefix = "\x19Ethereum Signed Message:\n32";

bytes32 prefixedHash = keccak256(abi.encodePacked(prefix, h));

address addr = ecrecover(prefixedHash, v, r, s);

return addr;

}

}