//SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

pragma experimental  ABIEncoderV2;

import "@openzeppelin/contracts/utils/Strings.sol";

contract SmartVaccineSystem

{

  address private owner;

  mapping(address=>sm) private sms;

  mapping(address=>sm)public slotmanagermapping;

  mapping (bytes32 => patient) private patients;

  mapping (address=> doctor) private doctors;

  uint count;

  uint private tcount;

  address[] public DoctorList;

  mapping(bytes32=>patient)public patientmapping;

  mapping(uint=>sSlot)public sSlotlist;

  mapping(address=>doctor)public doctormapping;

  mapping(uint=>bytes32)public pid;

  mapping(uint=>address)public did;

  mapping (bytes32 => bool) public isPatient;

  uint vaccinated\_count=2;

  string[] public CIDList;

  address[] SlotManagerList;

  struct patient {

      bool isuidgenerated;

      string name;

      uint8 age;

      string bloodgroup;

      string  image;

      bool isvaccinated;

      uint index;

      string status;

      string slotBooked\_at;

      string slotBooked\_on;

      string slotLocation;

    }

 struct doctor {

      string Doctor;

      uint Age;

      string Qualification;

      address id;

       address[] patient\_list;

  }

  struct sm{

      string slotmanager;

      address id;

      uint age;

  }

  struct sSlot{

      string loc;

      uint id;

      uint index;

      string slot1;

      string slot2;

      string slot3;

      uint s1\_c;

      uint s2\_c;

      uint s3\_c;

      uint s\_tc;

      uint vcount;

      uint vaccinated\_people\_count;

  }

 constructor() payable{

    owner = msg.sender;

  }

 modifier checkDoctor(address id) {

    doctor memory d = doctors[id];

    require(d.id > address(0x0));//check if doctor exist

    \_;

  }

modifier checkSlotManager(address id){

    sm memory m=sms[id];

    require(m.id > address(0x0));

    \_;

}

 modifier onlyOwner() {

    require(msg.sender == owner);

    \_;

  }

  function receive\_CID(string memory  cid) public{

    CIDList.push(cid);

  }

   function get\_CIDList() public view returns(string[] memory ){

        return CIDList;

    }

 function getSM() public view checkSlotManager(msg.sender) returns(string memory \_name, uint age){

    sm memory c = sms[msg.sender];

    return(c.slotmanager,c.age);

}

function getDoctor() public view checkDoctor(msg.sender) returns(string memory \_name,uint \_age, string memory \_qualification){

    doctor memory a = doctors[msg.sender];

    return(a.Doctor,a.Age,a.Qualification);

}

function SignupSlotManager(string memory \_name,uint age) public{

    sm memory m=sms[msg.sender];

    require(keccak256(abi.encodePacked(\_name))!=keccak256(""));

    require(!(m.id>address(0x0)));

    sms[msg.sender]=sm({slotmanager:\_name,id:msg.sender,age:age});

    slotmanagermapping[msg.sender].slotmanager=\_name;

    slotmanagermapping[msg.sender].age=age;

    slotmanagermapping[msg.sender].id=msg.sender;

    SlotManagerList.push(msg.sender);

  }

function get\_SlotManager\_List() public view returns(address[] memory ){

  return SlotManagerList;

}

function is\_SlotManager(address did) public view returns(bool value){

    for(uint i;i<SlotManagerList.length;i++){

      if(SlotManagerList[i]==did){

       return(true);

      }

    }

  }

function createMenu(uint index,string memory location,string memory slottime1,string memory slottime2,string memory slottime3,uint s1c,uint s2c,uint s3c) public checkSlotManager(msg.sender){

      sSlot storage place=sSlotlist[index];

      place.index=index;

      place.loc=location;

      place.slot1=slottime1;

      place.slot2=slottime2;

      place.slot3=slottime3;

      place.s1\_c=s1c;

      place.s2\_c=s2c;

      place.s3\_c=s3c;

      place.s\_tc=s1c+s2c+s3c;

      tcount=tcount+1;

  }

   function GetMenudetails(uint \_index) public view returns(uint i,string memory l,string memory s1,string memory s2,string memory s3,uint cs1,uint cs2,uint cs3){

    sSlot memory pp=sSlotlist[\_index];

     i=pp.index;

     l=pp.loc;

    s1=pp.slot1;

    s2=pp.slot2;

    s3=pp.slot3;

    cs1=pp.s1\_c;

    cs2=pp.s2\_c;

    cs3=pp.s3\_c;

    return(i,l,s1,s2,s3,cs1,cs2,cs3);

  }

  function Vaccine\_count(uint ind,uint count\_of\_vaccines)public  checkSlotManager(msg.sender)  {

  require(tcount!=0 && ind<tcount ,"Area not available");

  sSlot storage p=sSlotlist[ind];

  require(count\_of\_vaccines>=p.s\_tc);

  sSlotlist[ind].vcount=count\_of\_vaccines;

}

  function AvailabilityofVaccines(uint \_index) public view returns(uint vaccinecount){

    sSlot memory pp=sSlotlist[\_index];

     return(pp.vcount);

}

function vaccinatedareascount()public view checkSlotManager(msg.sender) returns(string[] memory)

{

   string[] memory arr=new string[](tcount);

   uint j=0;

   uint total\_people\_vaccinated=0;

  while(j<tcount){

     string memory area;

     uint i=sSlotlist[j].vaccinated\_people\_count;

     area=sSlotlist[j].loc;

    total\_people\_vaccinated+=i;

     string memory vparea=Strings.toString(i);

     string memory str=string(abi.encodePacked(area,"  -- >  ",vparea));

     arr[j]=str;

    j+=1;

   }

    return(arr);

   }

  function vaccinatedcount()public view checkSlotManager(msg.sender) returns(string[] memory)

{

   string[] memory arr=new string[](tcount);

   uint[] memory l=new uint[](tcount);

   sm memory m=sms[msg.sender];

   uint j=0;

   uint k=0;

   uint total\_people\_vaccinated=0;

   uint index=0;

  while(j<tcount){

     string memory area;

     uint i=sSlotlist[j].vaccinated\_people\_count;

     area=sSlotlist[j].loc;

     if(i<vaccinated\_count){

       l[k]=j;

       k=k+1;

     }

    total\_people\_vaccinated+=i;

    j+=1;

   }

   string memory stri;

   string[] memory location=new string[](k);

   for(uint z=0;z<k;z++){

    uint index=l[z];

    stri=sSlotlist[index].loc;

    location[z]=stri;}

    return location;

   }

  function registerDoctor(string memory \_name,uint age,string memory qualification) public{

    doctor memory d=doctors[msg.sender];

    require(keccak256(abi.encodePacked(\_name))!=keccak256(""));

    //require(age==age);

    require(!(d.id>address(0x0)));

    doctors[msg.sender]=doctor({Doctor:\_name,id:msg.sender,Age:age,Qualification:qualification,patient\_list:new address[](20)});

    doctormapping[msg.sender].Doctor=\_name;

    doctormapping[msg.sender].Age=age;

    doctormapping[msg.sender].Qualification=qualification;

    doctormapping[msg.sender].id=msg.sender;

     DoctorList.push(msg.sender);

  }

  function get\_Doctor\_List() public view returns(address[] memory ){return DoctorList;}

  function is\_Doctor(address did) public view returns(bool value){

    for(uint i;i<DoctorList.length;i++){

      if(DoctorList[i]==did){

       return(true);

      }

    }

  }

  function updatePatientDetails(bytes32 hash)public checkDoctor(msg.sender){

  require(keccak256(abi.encodePacked(patientmapping[hash].status))==keccak256(abi.encodePacked("Slot Booked")),"Slot Not Booked");

  require(patientmapping[hash].isvaccinated==false);

  patientmapping[hash].isvaccinated=true;

  string memory s=patientmapping[hash].slotLocation;// patient kommadi

  uint j=0;

  while(j<tcount)

  {

    if(keccak256(abi.encodePacked(s))==keccak256(abi.encodePacked(sSlotlist[j].loc))){

      require(sSlotlist[j].vcount!=0);

       sSlotlist[j].vcount-=1;

       sSlotlist[j].vaccinated\_people\_count+=1;

       break;

      }

      else {

        j+=1;

      }

    }

  }

  function signupPatient(string memory \_name, uint8 \_age,string memory \_bloodgroup)onlyOwner public {

      bytes32 uniqueId=sha256(abi.encodePacked(msg.sender,block.timestamp));

     require(!patientmapping[uniqueId].isuidgenerated);

     patientmapping[uniqueId].isuidgenerated=true;

     patientmapping[uniqueId].name=\_name;

     patientmapping[uniqueId].age=\_age;

     patientmapping[uniqueId].bloodgroup=\_bloodgroup;

     patientmapping[uniqueId].image="None"; // who generated address not patient address

     patientmapping[uniqueId].isvaccinated=false;

     patientmapping[uniqueId].slotBooked\_at="None";

     patientmapping[uniqueId].slotBooked\_on="None";

     patientmapping[uniqueId].status="Slot Not Booked";

     patientmapping[uniqueId].slotLocation="None";

     pid[count]=uniqueId;

     count=count+1;

     isPatient[uniqueId]=true;

  }

  function gethash(uint i) public view returns (bytes32 hash){

    return pid[i];

  }

  function patientdetails(bytes32 hash) public view returns(string memory name,uint8 age,string memory bloodgroup,string memory image,string memory status,string memory slotBooked\_on,string memory slotBooked\_at,string memory slotLocation,bool isvaccinated){

      patient memory a=patientmapping[hash];

      name=a.name;

      age=a.age;

      bloodgroup=a.bloodgroup;

      image=a.image;

      status=a.status;

      slotBooked\_on=a.slotBooked\_on;

      slotBooked\_at=a.slotBooked\_at;

      slotLocation=a.slotLocation;

      isvaccinated=a.isvaccinated;

     return(name,age,bloodgroup,image,status,slotBooked\_on,slotBooked\_at,slotLocation,isvaccinated);

  }

  function uploadimage(bytes32 hash, string memory image) public{

      patientmapping[hash].image=image;

  }

function BookSlot(bytes32 hash,uint \_index,uint SlotId,string memory slotBooked\_on) public{

      require(keccak256(abi.encodePacked(patientmapping[hash].status))==keccak256(abi.encodePacked("Slot Not Booked")),"Slot Booked ");

    sSlot memory pp=sSlotlist[\_index];

    require(sSlotlist[\_index].vcount!=0);

    require(sSlotlist[\_index].s\_tc!=0);

    patientmapping[hash].slotLocation=sSlotlist[\_index].loc;

    if(SlotId==1){

    require(sSlotlist[\_index].s1\_c!=0);

     patientmapping[hash].slotBooked\_at=pp.slot1;

     patientmapping[hash].status="Slot Booked";

    patientmapping[hash].slotBooked\_on=slotBooked\_on;

    sSlotlist[\_index].s\_tc-=1;

    sSlotlist[\_index].s1\_c-=1;

    }

    else if(SlotId==2){

     require(sSlotlist[\_index].s2\_c!=0);

     patientmapping[hash].slotBooked\_at=pp.slot2;

     patientmapping[hash].status="Slot Booked";

    patientmapping[hash].slotBooked\_on=slotBooked\_on;

     sSlotlist[\_index].s\_tc-=1;

    sSlotlist[\_index].s2\_c-=1;

    }

    else if(SlotId==3){

    require(sSlotlist[\_index].s3\_c!=0);

    patientmapping[hash].slotBooked\_at=pp.slot3;

    patientmapping[hash].status="Slot Booked";

    patientmapping[hash].slotBooked\_on=slotBooked\_on;

    sSlotlist[\_index].s\_tc-=1;

    sSlotlist[\_index].s3\_c-=1;

    }

    else{

      patientmapping[hash].slotBooked\_at="None";

      patientmapping[hash].status="Slot Not Booked";

    }

  }

  function CancelSlot(bytes32 hash,uint \_index,uint slotId ) public{

    require(keccak256(abi.encodePacked(patientmapping[hash].status))==keccak256(abi.encodePacked("Slot Booked")),"Slot Not Booked ");

    require(patientmapping[hash].isvaccinated==false);

      sSlotlist[\_index].s\_tc+=1;

      if(slotId==1){

       sSlotlist[\_index].s1\_c+=1;

       patientmapping[hash].slotBooked\_at="None";

        patientmapping[hash].slotBooked\_on="None";

      }

      else if(slotId==2){

        sSlotlist[\_index].s2\_c+=1;

        patientmapping[hash].slotBooked\_at="None";

         patientmapping[hash].slotBooked\_on="None";

      }

      else if(slotId==3){

        sSlotlist[\_index].s3\_c+=1;

        patientmapping[hash].slotBooked\_at="None";

         patientmapping[hash].slotBooked\_on="None";

      }

      patientmapping[hash].status="Slot Not Booked";

      patientmapping[hash].slotLocation="None";

       patientmapping[hash].slotBooked\_on="None";

  }

}s