Admin.sol

```
pragma solidity ^0.8.0;


contract Admin {

    //add patient

    //add doctor

    //view patient

    //view doctor

    //authenticator


    //declare patient and doctor using structures.

    struct admin{

        string admin_name;

        int admin_age;

        string admin_mail;

        string admin_address;

    } admin a;


    mapping(int => admin) all_admins;   //all different 'p' will be
stored in all_patients.

    int a_len = 0; //to keep track of mapping length. Usse for loop ko
chalayenge.


    struct doctor{

        string doctor_name;
        int doctor_age;

        string doctor_mail;

    } doctor d;


    mapping(int => doctor) all_doctors;

    int d_len = 0;


    address owner;

    //only admin can modify data

   modifier onlyOwner() {

        require(msg.sender == owner);

        _;

    }


    constructor() public {

        owner = msg.sender;

    }

//store value in p(store patient)

    function store_value_in_a(int a_id, string memory a_name, int
a_age, string memory a_mail, string memory a_address) public {

        a.admin_name = a_name;

        a.admin_age = a_age;

        a.admin_mail = a_mail;

        a.admin_address = a_address;


        a_len+=1;
        all_admins[a_id] = a;

    }


    //access patients index by index

    function admin_by_admin(int a_id) public view returns(string
memory, int, string memory,string memory){

    admin memory a = all_admins[a_id];

    return (a.admin_name, a.admin_age, a.admin_mail,a.admin_address);

    }


    //view all patients in system

    function view_admins() public view  {


        for(int i = 0; i<a_len; i++){

           admin_by_admin(i);

        }

    }


    //store value in d(store doctor)

    function store_value_in_d(int d_id, string memory d_name, int
d_age, string memory d_mail) public {

        d.doctor_name = d_name;

        d.doctor_age = d_age;

        d.doctor_mail = d_mail;
        d_len+=1;



        all_doctors[d_id] = d;

    }







    //access doctors index by index

    function doctor_by_doctor(int d_id) public view returns(string
memory, int, string memory){

    doctor memory dt = all_doctors[d_id];

 return (dt.doctor_name, dt.doctor_age, dt.doctor_mail);

    }


    //view all doctors in system

    function view_doctors() public view  {

         for(int i = 0; i<d_len; i++){

         doctor_by_doctor(i);

        }

    }



}
```

Patient.sol

```
pragma solidity ^0.8.0;


contract Patient {
    //add patient
    //add doctor
    //view patient
    //view doctor
    //authenticator


    //declare patient and doctor using structures.
    struct patient{
        string patient_name;
        int patient_age;
        string patient_mail;


    } patient p;      //instance for every patient.


    mapping(int => patient) all_patients;   //all different 'p' will be
stored in all_patients.
    int p_len = 0; //to keep track of mapping length. Usse for loop ko
chalayenge.




    address owner;
    //only admin can modify data

   modifier onlyOwner() {

        require(msg.sender == owner);

        _;

    }



    constructor() public {

//address of admin

        owner = 0x548Fa45eae0F18dE499776bE7A4DEaE438162d71;

    }

//store value in p(store patient)

    function store_value_in_p(int p_id, string memory p_name, int
p_age, string memory p_mail) public {

        p.patient_name = p_name;

        p.patient_age = p_age;

        p.patient_mail = p_mail;





        p_len+=1;



        all_patients[p_id] = p;

    }







    //access patients index by index
    function patient_by_patient(int p_id) public view returns(string
memory, int, string memory){

    patient memory pt = all_patients[p_id];

    return (pt.patient_name, pt.patient_age, pt.patient_mail);

    }


    //view all patients in system

    function view_patients() public view  {


        for(int i = 0; i<p_len; i++){

           patient_by_patient(i);

        }

    }


    struct history{

        string hcond1;

        string hcond2;

    }history h;     //instance for every prescription.


    mapping(int => history) all_history;


    function store_value_in_h(int h_id, string memory h_cond1, string
memory h_cond2) public  {

        h.hcond1 = h_cond1;

        h.hcond2 = h_cond2;
        all_history[h_id] = h;

    }



    function history_by_history(int h_id) public view returns(string
memory, string memory){

    history memory h = all_history[h_id];

    return (h.hcond1 , h.hcond2);

    }






}
```

Doctor_Ex.sol

```
pragma solidity ^0.8.0;


import 'contracts/patient.sol';










 import
"https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/con
tracts/token/ERC721/ERC721.sol";


contract Doctor is ERC721{

    //view patient list

    //add diagnosis

    //add prescription

    //authenticator






    address owner;


    constructor() ERC721("Doctor_Token", "D_T") public{

        owner = msg.sender; //address of doctor generated after
deploying.
```

```
    }


    //only doctor can modify data

   modifier onlyOwner() {

        require(msg.sender == owner);

        _;

    }







    /*function view_patient_list(int p_id) public view returns(string
memory, int, string memory){

        address admin = 0x7d12c01C4D182F62bC81B45DAa6D976D5288Ba31;

        Admin a = Admin(admin);

        return a.patient_by_patient(p_id);

    }*/


    function view_patient_list(int id) public view returns(string
memory, int, string memory){

        address patient = 0xFdf39f5606CCe459F0A5E9F380941A4A6EF630E0;

        Patient pt = Patient(patient);

        return pt.patient_by_patient(id);

    }
    function view_patient_history(int id) public view returns(string
memory, string memory){

        address history = 0xFdf39f5606CCe459F0A5E9F380941A4A6EF630E0;

        Patient h = Patient(history);

        return h.history_by_history(id);

    }


     struct diagnosis{

        string diagnosis_name;

    }diagnosis dia;     //instance for every diagnosis.


    mapping(int => diagnosis) all_diagnosis;


     struct prescription{

        string med1;

        string med2;

    }prescription pres;     //instance for every prescription.


    mapping(int => prescription) all_prescriptions;


    //store value in dia

    function store_value_in_dia(int dia_id, string memory dia_name)
public {

        dia.diagnosis_name = dia_name;
        all_diagnosis[dia_id] = dia;

    }



    //store value in pres

    function store_value_in_pres(int pres_id, string memory pres_med1,
string memory pres_med2) public {

        pres.med1 = pres_med1;

        pres.med2 = pres_med2;



        all_prescriptions[pres_id] = pres;

    }



    function view_diagnosis(int dia_id) public view returns(string
memory){

        diagnosis memory dia_obj = all_diagnosis[dia_id];



        return dia_obj.diagnosis_name;

    }



    function view_prescription(int pres_id) public view returns(string
memory, string memory){

        prescription memory pres_obj = all_prescriptions[pres_id];



        return (pres_obj.med1, pres_obj.med2);

    }
    /*function view_value_in_h(int id) public view returns(string
memory,string memory)

    {

        address patient = ;

        Patient p = Patient(patient);

        return (p.h_cond1,p.h_cond2);

    }*/




}
```

Insurance.sol

```
pragma solidity ^0.8.2;

contract Insurance {

    address Admin;

    struct patient{
        int id;
        address uid;
        string name;
        int amountInsured;
    }

    mapping(int => patient) public patientmapping;
    mapping(address => patient) public patientclaiming;

    constructor(){
     Admin = msg.sender;
    }

    modifier onlyOwner(){
        require(Admin == msg.sender);
        _;
    }


    function setpatientData(int id, string memory _name, int
_amountInsured) public onlyOwner{
        address uniqueID =
address(bytes20(sha256(abi.encodePacked(msg.sender,
'block.timestamp'))));
        patientmapping[id].uid = uniqueID;
        patientmapping[id].name = _name;
        patientmapping[id].amountInsured = _amountInsured;

        patientclaiming[uniqueID].id = id;
        patientclaiming[uniqueID].name = _name;
        patientclaiming[uniqueID].amountInsured = _amountInsured;
    }
    function useInsurance(int id, address _uniqueID, int _amountUsed)
public onlyOwner returns (string memory){
        if(patientclaiming[_uniqueID].amountInsured < _amountUsed){
            return "Error";
        }
        patientclaiming[_uniqueID].amountInsured -= _amountUsed;
        patientmapping[id].amountInsured -= _amountUsed;
        return "Insurance used successfully";

    }

}
```

Manu_reg.sol

```
pragma solidity ^0.8.0;

contract Manu_Reg {
    //add manu
    //add doctor
    //view manu
    //view doctor
    //authenticator

    //declare manu and doctor using structures.
    struct manu{
        string manu_name;
        int manu_age;
        string manu_mail;

    } manu p;      //instance for every manu.

    mapping(int => manu) all_manus;   //all different 'p' will be
stored in all_manus.
    int p_len = 0; //to keep track of mapping length. Usse for loop ko
chalayenge.



    address owner;
    //only admin can modify data
   modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    constructor() public {
//address of admin
        owner = 0xF1182D18C07738189103bad61D40E73Cd6cCA7D2;
    }
//store value in p(store manu)
    function store_value_in_p(int p_id, string memory p_name, int
p_age, string memory p_mail) public {
        p.manu_name = p_name;
        p.manu_age = p_age;
        p.manu_mail = p_mail;
        p_len+=1;

        all_manus[p_id] = p;
    }



    //access manus index by index
    function manu_by_manu(int p_id) public view returns(string memory,
int, string memory){
    manu memory pt = all_manus[p_id];
    return (pt.manu_name, pt.manu_age, pt.manu_mail);
    }
}
```

Distributor.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import
"@openzeppelin/contracts@4.4.2/token/ERC721/IERC721Receiver.sol";

contract Distributer is IERC721Receiver{

    function onERC721Received(address operator,address from,uint256
tokenId,bytes calldata data) external override returns (bytes4){
        return this.onERC721Received.selector;
    }


}
```

Manufacturer.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts@4.4.2/token/ERC721/ERC721.sol";
import
"@openzeppelin/contracts@4.4.2/token/ERC721/extensions/ERC721Enumerable
.sol";
import "@openzeppelin/contracts@4.4.2/access/Ownable.sol";
import "@openzeppelin/contracts@4.4.2/utils/Counters.sol";

contract Manufacturer is ERC721, ERC721Enumerable, Ownable {
    using Counters for Counters.Counter;

    Counters.Counter private _tokenIdCounter;

    constructor() ERC721("MyNFT", "MNFT") {}

    function _baseURI() internal pure override returns (string memory)
{
        return
"https://my-json-server.typicode.com/abcoathup/samplenft/tokens/";
    }

    function safeMint(address to) public {
        uint256 tokenId = _tokenIdCounter.current();
        _tokenIdCounter.increment();
        _safeMint(to, tokenId);
    }

    // The following functions are overrides required by Solidity.

    function _beforeTokenTransfer(address from, address to, uint256
tokenId)
        internal
        override(ERC721, ERC721Enumerable)
    {
        super._beforeTokenTransfer(from, to, tokenId);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721Enumerable)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```