CSE 528
Introduction to Blockchain and Cryptocurrency

Group Project Progress Report-3

Team Members :

Aabhaas Batra	2019001
Raj Kumar	2019084
Udit Narang	2019120

Problem Statement:

We have seen significant developments in medical facilities over the years.
But from the perspective of visitors or patients at hospitals, clinics, there are still some procedures that are inefficient and cause discomfort to the citizens.People always need to carry their medical records whenever they see the doctor. And it’s tough to keep a record of all your documents and take them with you each time you go to the hospital or clinic.

Idea:

Our idea is to provide a solution to this problem using blockchain technology. We aim to store people’s medical records on the blockchain, which will keep the information safe and accessible. This would also help us in decentralizing the data in the medical field.
The primary users of our network would be the doctors and the patients.
The patient will have the authority to give access to their records to anyone. And hence, it would be possible to retrieve a patient’s medical history from any hospital in the world.

Planned Goals

03 Nov 2021 - 21 Nov 2021 :

- Designing the nodes of our blockchain
- Designing the blockchain instance
- Building the required APIs

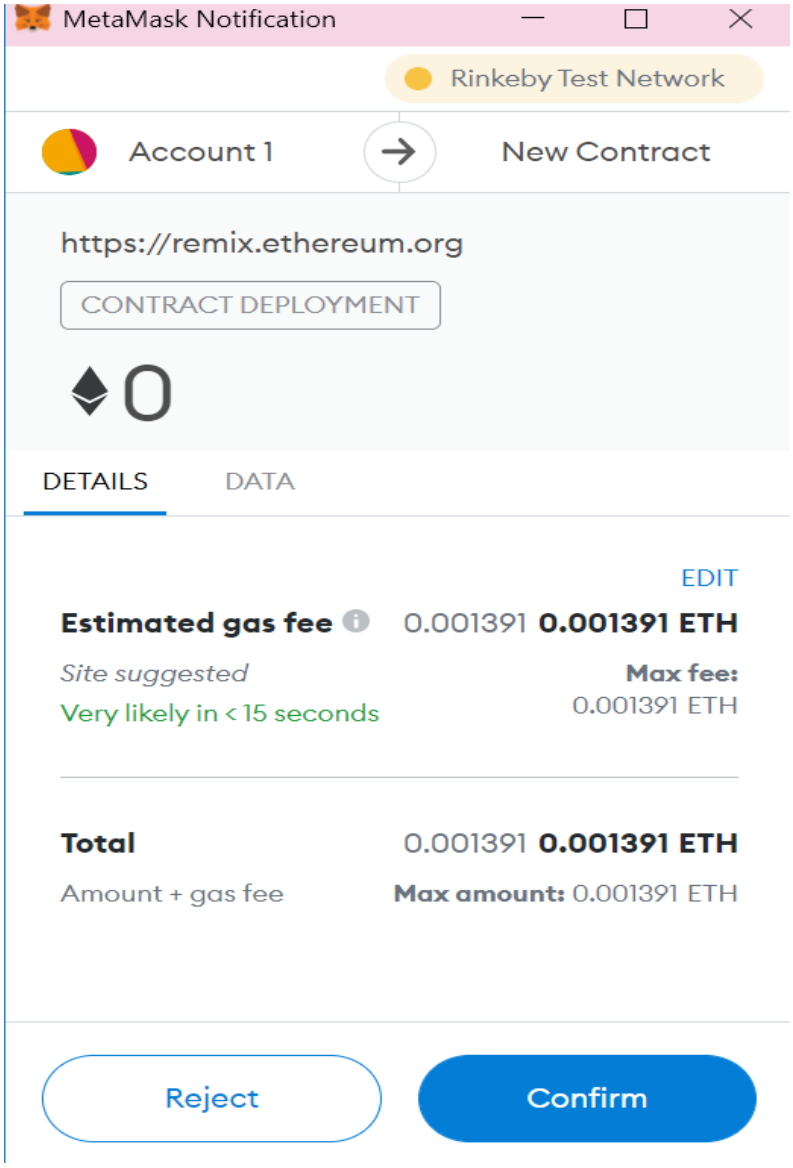
Individual Contributions by each team member for weeks 5 and 6 are as follows:

Code for Hospital Node

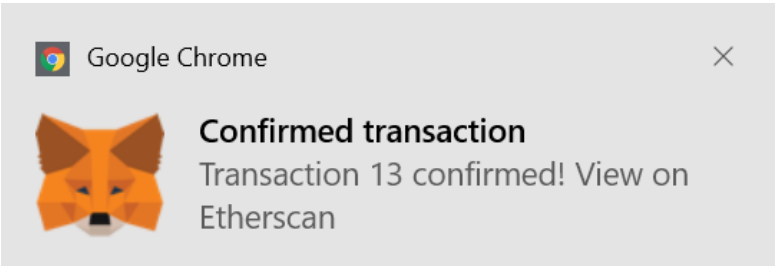
```
pragma solidity >= 0.4.22 <0.7.0;

contract Hospital {

    mapping(uint256 => hospital) h_list;

    address h_owner;

    constructor() public{
        h_owner = msg.sender;
    }

    modifier isH_Owner(){                       /*to maintain hospital's access
only*/

        require(msg.sender == h_owner, "Access Denied");
        _;
    }

    struct hospital{

        string h_name;
        string h_address;
        string h_field;   /* Specialization*/
        string h_yoe;     /* Year of establishment*/
    }
    hospital _hospital;

    function getHospital(uint256 h_id) public view returns(string memory, string
memory, string memory, string memory)
    {
        hospital memory hospital_ = h_list[h_id];
        return (hospital_.h_name, hospital_.h_address, hospital_.h_field,
hospital_.h_yoe);
    }

    function addHospital(string memory h__name, string memory h__address, string
memory h__field, string memory h__yoe, uint256 h_id)public isH_Owner{

        _hospital.h_name = h__name;
        _hospital.h_address = h__address;
```

```
        _hospital.h_field = h__field;
        _hospital.h_yoe = h__yoe;
        h_list[h_id] = _hospital;
    }
}
```

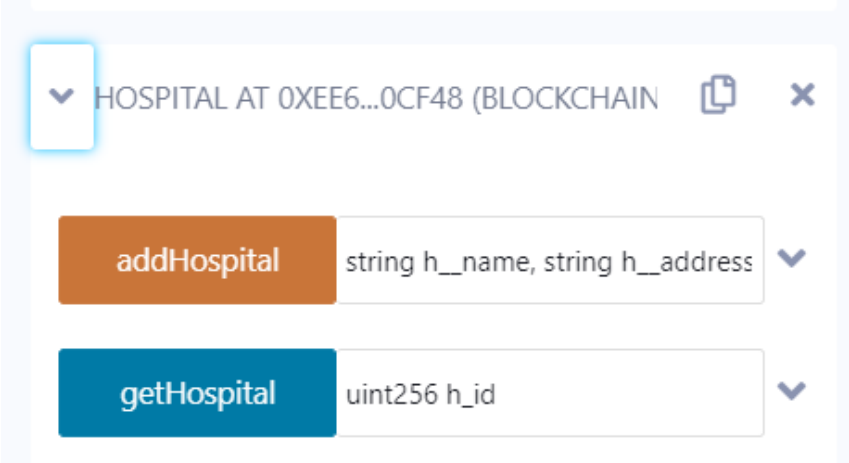
Deploying hospital.sol using Metamask Account



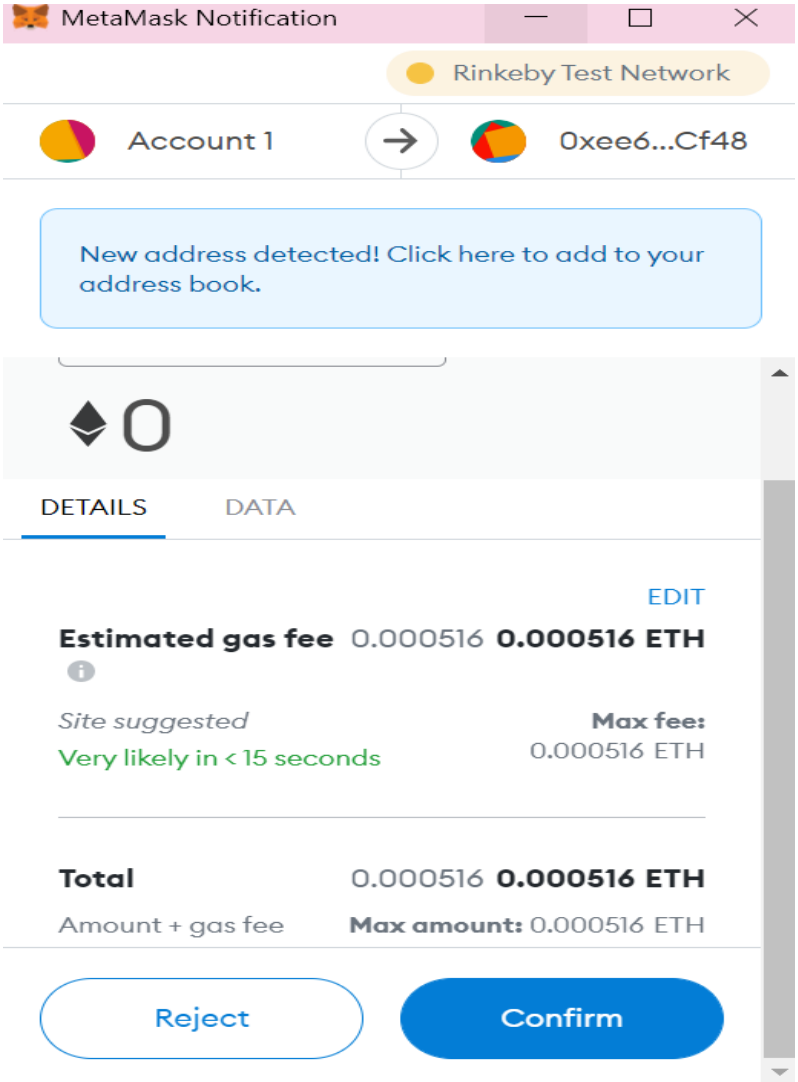
Confirmation of Deployment success



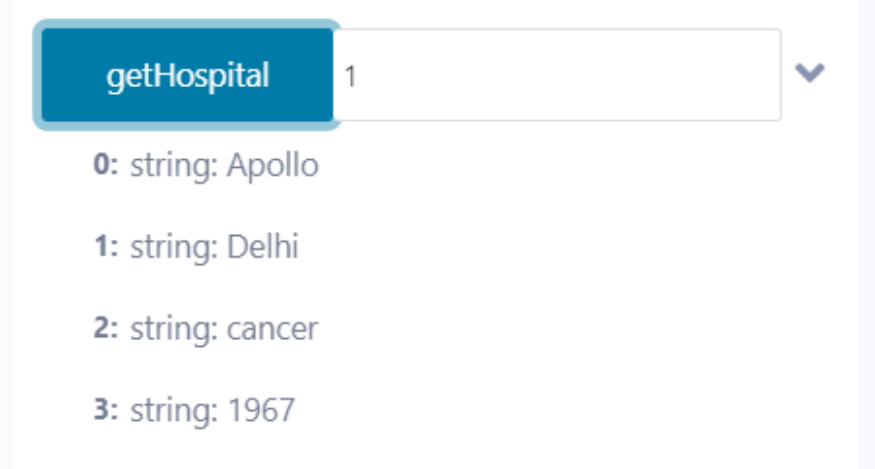
Hospital Node successfully created



Calling setGuardianDetails function using Metamask account



Displaying the Hospital details using getHospital function



Code for records.sol

```
pragma solidity >= 0.4.22 <0.7.0;

import
"https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/tok
en/ERC721/ERC721.sol";

contract Records is ERC721{

    struct patient{

        uint256 p_id;
    }
    patient _patient;


    struct patient_history{

        uint256 p_id;
        string pers_hist;
        string family_hist;

    }
    patient_history hist;

    struct present_issue{

        uint256 p_id;
        string issue;
```

```
    string visits;
    string startDate;
    string duration;

}
present_issue pres_issue;

struct diagnosis{

    uint256 p_id;
    string issue;
    string presc;
}
diagnosis _diagnosis;

struct admission{

    uint256 doc_id;
    uint256 h_id;
    string issue;
    string adm_date;
    string discharge_date;
    string presc;
    string follow_up;
}
admission adm;


struct previous_dates{

    uint256 p_id;
    string _previous_dates;
}
previous prev_dates;

struct insurance{

    uint256 p_id;
    uint64 pol_num;
    string pol_type;
    string pol_lim;
    string applicable;
    string insurer;
}
insurance _insurance;
```

```
    mapping(uint256 => patient) p_list;
    mapping(uint256 => patient_history) p_history;
    mapping(uint256 => present_issue) p_issue;
    mapping(uint256 => diagnosis) diag_list;
    mapping(uint256 => admission) adm_list;
    mapping(uint256 => previous_dates) prev_list;
    mapping(uint256 => insurance) i_list;

    address r_owner;

    constructor() ERC721("RaabudCoin","RBC") public {

          /*This has to be changed later on. This is just an example value*/
          r_owner = 0x34d8bC94989BbE14BCfd98E0550201ba4970B776; //Address of
Doctor

    }

    modifier isR_Owner(){                       /*to maintain doctor's access
only*/

        require(msg.sender == r_owner, "Access Denied");
        _;
    }

    //functions related to the RBC coin
    function getName() public view returns(string memory)
    {
        return name();
    }

    function getSymbol() public view returns(string memory)
    {
        return symbol();
    }

    function getTotalCOunt() public view returns(uint256)
    {
        return totalSupply();
    }


    //function for minting the RBC coin
    function med_rec(uint256 p__id) public{
```

```
        _mint(msg.sender, p__id);
        p_list[p__id] = _patient;
    }


    //functions for patient medical history
    function getPatientHistory(uint256 p__id)public view returns (string
memory,string memory){

        patient_history memory p_h = p_history[p__id];
        return (p_h.personal_history, p_h.family_history);
    }

    function addPatientHistory(uint256 p__id, string memory pers__hist, string
memory family__hist)public isR_Owner {

        hist.pers_hist = pers__hist;
        hist.family_hist = family__hist;
        p_history[p__id] = hist;
    }


    //functions for present issues
    function getPresentIssue(uint256 p__id)public view returns (string
memory,string memory, string memory, string memory){

        present_issue memory pr_is = p_issue[p__id];
        return (pr_is.issue, pr_is.visits, pr_is.startDate, pr_is.duration);
    }

    function addPresentIssue(uint256 p__id,string memory _issue,string memory
_visits, string memory _startDate, string memory dur)public isR_Owner {

        pres_issue.issue = _issue;
        pres_issue.visits = _visits;
        pres_issue.startDate = _startDate;
        pres_issue.duration = dur;
        p_issue[p__id] = pres_issue;
    }


    //functions for diagnosis of a patient
    function get_func_diagnosis(uint256 p__id)public view returns (string
memory,string memory){

        diagnosis memory diag = diag_list[p__id];
```

```
        return (diag.issue, d.presc);

    }

    function addDiagnosis(uint256 p__id,string memory _issue,string memory
_presc)public isR_Owner {

        _diagnosis.issue = _issue;
        _diagnosis.presc = _presc;
        diag_list[p__id] = _diagnosis;

    }


    //functions for admission of a patient
    function getAdmisssion(uint256 p__id)public view returns (uint256,uint256,
string memory, string memory, string memory ,string memory, string memory){

        admission memory ad = admission[p___id];
        return (ad.doc_id, ad.h_id, ad.issue, ad.adm_date, adm.discharge_date,
adm.presc, adm.follow_up);

    }

    function addAdmission(uint256 p__id, uint256 doc__id, uint256 h__id, string
memory _issue, string memory adm__date,string memory discharge__date, string
memory _presc, string memory follow__up)public isR_Owner {

        adm.doc_id = _doc__id;
        adm.h_id = h__id;
        adm.issue = _issue;
        adm.adm_date = adm__date;
        adm.discharge_date = discharge__date;
        adm.presc = _presc;
        adm.follow_up = follow__up;
        adm_list[p__id] = adm;

    }


    //fuctions for previous dates
    function getPrevDates(uint256 p__id) public view returns(string memory){

        previous_dates memory p_d = prev_list[p__id];
        return(p_d._previous_dates);
    }
```

```
    function addPrevDates(uint256 p__id, string memory p_dates) public isR_Owner{

        prev_dates._previous_dates = p_dates;
        prev_list[p__id] = prev_dates;
    }



    //functions for insurance
    function getInsurance(uint256 p__id)public view returns (uint64, string
memory, string memory,string memory,string memory){

        insurance memory insu = i_list[p__id];
        return (insu.pol__num, insu.pol__type, insu.pol__lim, insu.applicable,
insu.insurer);
    }

    function addInsurance( uint256 p__id, uint64 pol__num, string memory
pol__type, string memory pol__lim, string memory app, string memory ins)public
isR_Owner
    {


        _insurance.pol_num = pol__num;
        _insurance.pol_type = pol__type;
        _insurance.pol_lim = pol__lim;
        _insurance.applicable = _app;
        _insurance.insurer = ins;
        i_list[p__id] = i;
    }
}
```

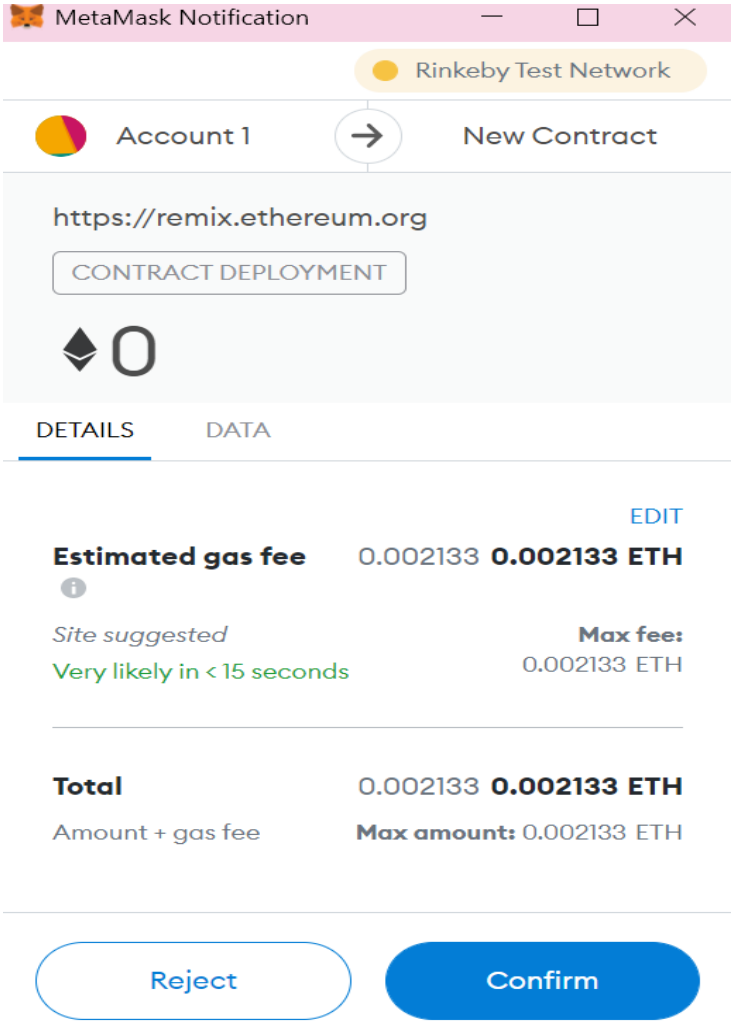
Code for patient node:

```
pragma solidity >=0.4.22 <0.7.0;

contract Patient
{
    struct Patient
    {
        uint256 patientID;
        string patientName;
        uint256 age;
        string gender;
        uint256 height;
        uint256 weight;
        string emailID;
        uint256 phoneNumber;
        uint256 doctorID;
        uint256 hospitalID;
        uint256 date;
    }

    struct Guardian
    {
        string guardianName;
        uint256 patientID;
        string relationWithPatient;
        uint256 phoneNumber;
        string emailID;
    }

    mapping(uint256 => Patient) patientList;
    mapping(uint256 => Guardian) guardianList;

    Patient p;
    Guardian g;

    address owner;

    constructor()  public
    {
        owner=0xc9571c3DF94Cb9BA3ed3ac32AaF47DBfD9E19bA9;
    }
```

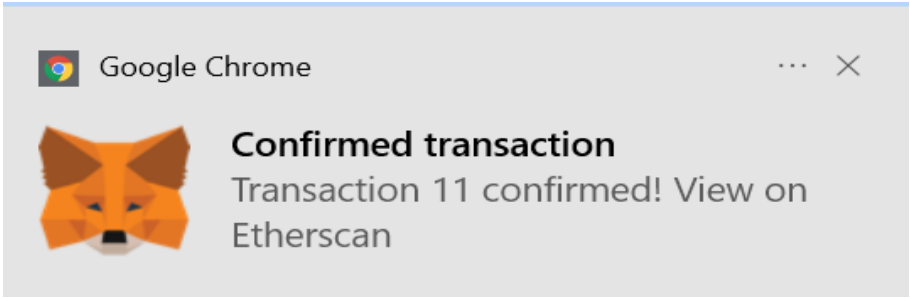
```
    modifier isOwner()
    {
        require(msg.sender == owner, "Access is not allowed");
         _;
    }

    function setPatientDetails(uint256 patientID,string memory patientName,
uint256 age, string memory gender,uint256 weight,uint256 height,string memory
emailID, uint256 phoneNumber, uint256 date) public isOwner
    {
        p.patientID=patientID;
        p.patientName=patientName;
        p.age=age;
        p.gender=gender;
        p.weight=weight;
        p.height=height;
        p.emailID=emailID;
        p.phoneNumber=phoneNumber;
        p.date=date;
    }

    function setGuardianDetails(uint256 patientID,string memory
guardianName,string memory relationWithPatient, uint256 phoneNumber, string
memory emailID) public isOwner
    {

        g.patientID = patientID;
        g.guardianName=guardianName;
        g.relationWithPatient=relationWithPatient;
        g.phoneNumber=phoneNumber;
        g.emailID=emailID;
        guardianList[patientID] = g;
     }

    function getPatientDetails(uint256 patientID) public view returns (uint256,
string memory, uint256, string memory,uint256, uint256, string memory, uint256,
uint256)
    {
        Patient memory p1 = patientList[patientID];
        return (p1.patientID, p1.patientName,p1.age, p1.gender, p1.height,
p1.weight, p1.emailID, p1.phoneNumber,p1.date);
    }

    function getGuardianDetails(uint256 patientID) public view returns(string
memory,string memory, uint256, string memory)
    {
```

```
        Guardian memory g1=guardianList[patientID];
        return (g1.guardianName, g1.relationWithPatient, g1.phoneNumber,
g1.emailID);
    }
}
```

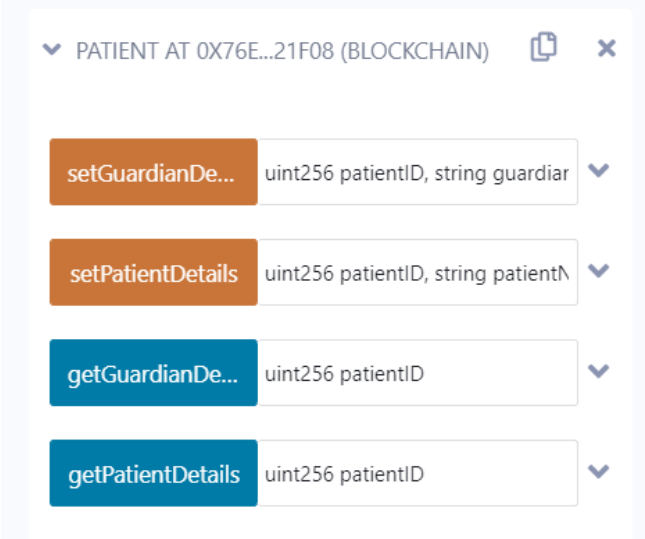
Deploying patient.sol using Metamask Account



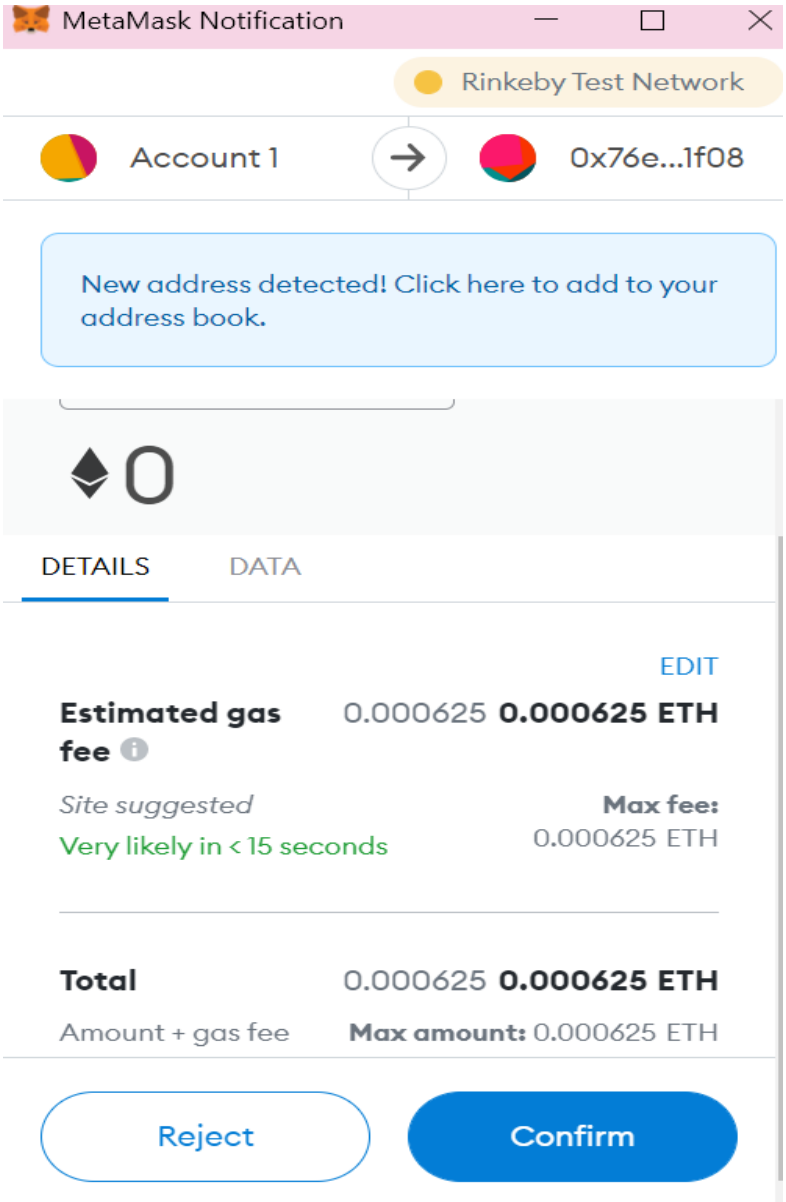
Confirmation of Deployment success



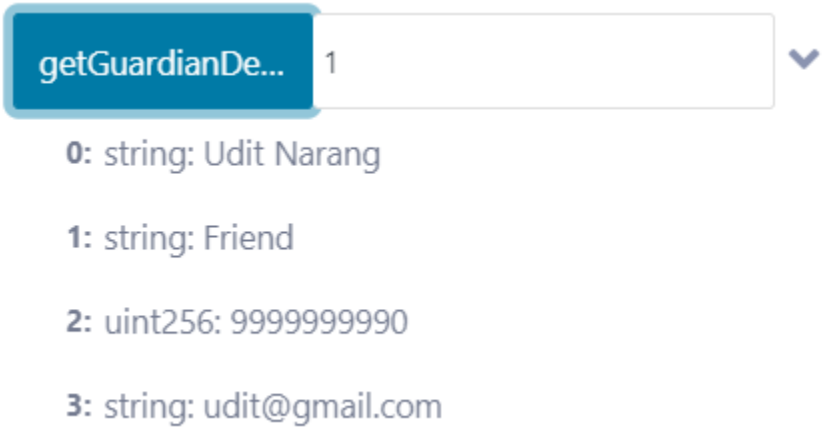
Patient Node successfully created



Calling setGuardianDetails function using Metamask account



Displaying the updated guardian details using getGuardianDetails function



Code for myBodyExamine.sol

```
pragma solidity >= 0.4.22 <0.7.0;
import
"https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/tok
en/ERC721/ERC721.sol";

contract bodyExamine is ERC721
{
    mapping(uint256=> bodyTests) myBodyTests;
    mapping(uint256=> bodyScans) myBodyScans;
    mapping(uint256=> system) systemExamine;
    mapping(uint256=> prev) prevDates;
    mapping(uint256=> Patient) patientList;

    struct Patient
    {
        uint256 patientID;
    }

    Patient p;

    struct prev
    {
        uint256 patientID;
        string previous;
    }

    prev pr;
```

```
struct bodyTests
{
    uint256 patientID;
    string bloodTest;
    string urineTest;
    string ecg;
    string mriScan;
    string ctScan;
    string xRay;
    string labTest;
}

bodyTests t;

struct bodyScans
{
    uint patientID;
    string built;
    string eyes;
    string tongue;
    uint256 pulse;
    uint256 temperature;
    string bloodPressure;
    uint256 respiratoryRate;
}

bodyScans s;

struct system
{
    uint64 patientID;
    string cns;
    string cvs;
    string rs;
    string abdomen;
}

system sys;

address owner;


constructor() ERC721("MedicalCoin","MEDC") public
{
    owner = 0xc9571c3DF94Cb9BA3ed3ac32AaF47DBfD9E19bA9;
}
```

```
    modifier isOwner()
    {
        require(msg.sender == owner, "Access is not allowed");
        _;
    }


    //built-in functions in ERC721

    function namedecl() public view  returns (string memory)
    {
        return name();
    }

    function symboldecl() public view  returns (string memory)
    {
        return symbol();
    }

    function totalSupplycount() public view  returns (uint256)
    {
        return totalSupply();
    }

    function medical_record(uint256 patientID) public
    {
        _mint(msg.sender,patientID);
        patientList[patientID] = p;
    }




    function previous_dates(uint256 patientID,string memory _previous)public
isOwner
    {
        pr.previous = _previous;
        prevDates[patientID] = pr;
    }

    function get_previous_dates(uint256 patientID)public view returns (string
memory)
    {
        prev memory pr1 = prevDates[patientID];
        return (pr1.previous);
```

```
    }


    function investigations(uint256 patientID,string memory _blood_test,string
memory _urine_test,string memory _ecg,string memory _mri_scan,string memory
_ct_scan,string memory _xray,string memory _lab_test)public isOwner
    {
        t.bloodTest = _blood_test;
        t.urineTest = _urine_test;
        t.ecg = _ecg;
        t.mriScan = _mri_scan;
        t.ctScan = _ct_scan;
        t.xray = _xray;
        t.labTest = _lab_test;
        myBodyTests[patientID] = t;
    }

    function get_investigations(uint256 patientID)public view returns (string
memory,string memory,string memory,string memory,string memory,string
memory,string memory)
    {
        bodyTests memory t1 = myBodyTests[patientID];
        return (t1.bloodTest, t1.urineTest, t1.ecg, t1.mriScan, t1.ctScan,
t1.xRay, t1.labTest);
    }

    function general_examin(uint256 patientID,string memory _built,string memory
_eyes,string memory _tongue,uint64 _pulse,string memory _blood_pressure,uint64
_temp,uint64 _respiratory_rate)public isOwner
    {
        s.built = _built;
        s.eyes = _eyes;
        s.tongue = _tongue;
        s.pulse = _pulse;
        s.bloodPressure = _blood_pressure;
        s.temperature = _temp;
        s.respiratoryRate = _respiratory_rate;
        myBodyScans[patientID] = s;
    }

    function get_general_examin(uint256 patientID)public view returns (string
memory, string memory, string memory, uint64, string memory, uint256, uint256)
    {
        bodyScans memory s1 = myBodyScans[patientID];
        return (s1.built, s1.eyes, s1.tongue, s1.pulse, s1.bloodPressure,
s1.temperature, s1.rrespiratoryRate);
```

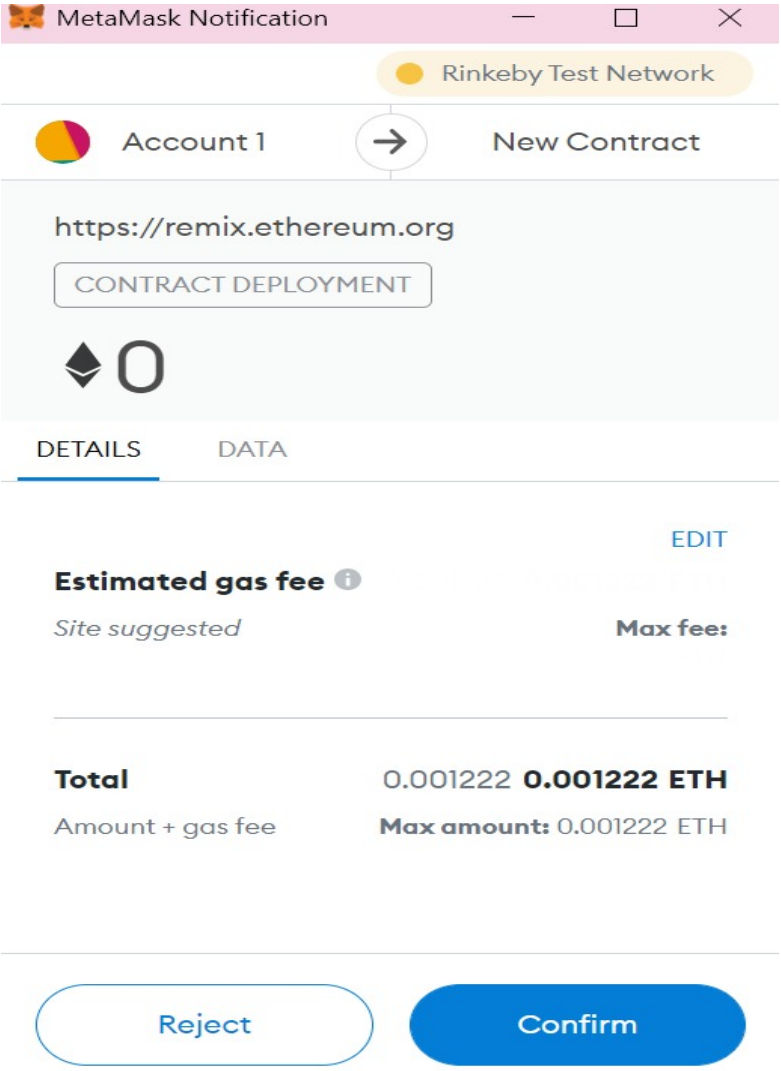
```
    }


    function sys_examin(uint256 patientID, string memory _cvs, string memory
_cns, string memory _rs, string memory _abdomen)public isOwner
    {
        sys.cvs = _cvs;
        sys.cns = _cns;
        sys.rs = _rs;
        sys.abdomen = _abdomen;
        systemExamine[patientID] = sys;
    }

    function get_sys_examin(uint256 patientID) public view returns (string
memory, string memory, string memory, string memory)
    {
        system memory sys1 = systemExamine[patientID];
        return (sys1.cvs, sys1.cns, sys1.rs, sys1.abdomen);
    }
}
```

Code for Doctor.sol:

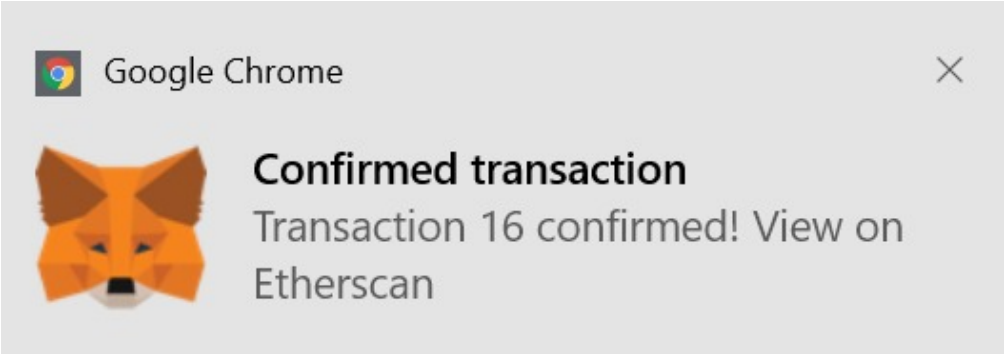
```
pragma solidity >=0.4.22 < 0.7.0;
/**
 * @title Medical records
 * @dev Store & retreive Doctor details
 */
contract Doctor {
     struct doctor{
         string name;
         string main_specialization;
         uint256 phone_no;
         string doct_address;
     }

      mapping(uint256 => doctor) list_of_doctors;//map that will have <id,doctor>
as key,value pair where id is of uint256 datatype

     doctor d;
     address owner;
      constructor()  public {
          owner = 0xE6005Cc724c2d44F0aF23d663017a7E375DD7F35; //Address of
Hospital
      }

     // modifier to give access only to hospital
       modifier isOwner() {

         require(msg.sender == owner, "Access is not allowed");
         _;
     }

     /*
     This function adds details of doctor to the list_of_doctors
     It takes id,doctor address,doctor name,doctor specialization as arguements
     This updates map list_of_doctors which contains <doctor_id,doctor> as  <key
, value> pair
     */
     function add_details_of_doctor(uint256 doctor_id,string memory
doct_address,uint256 phone_no,string memory name,string memory
specialization)public isOwner{
         d.name=name;
         d.phone_no=phone_no;
```

```
        d.main_specialization=specialization;
        d.doct_address=doct_address;
        list_of_doctors[doctor_id] = d;
    }

    /*
    This function returns details of doctor from the list_of_doctors
   It takes doctor_id as arguement which is already present as a key in the map
list_of_doctors
    */

   function retreive_details_of_doctor(uint16 id) public view returns (string
memory,string memory,uint256,string memory){

       doctor memory dp = list_of_doctors[id];
       return (dp.name,dp.main_specialization,dp.phone_no,dp.doct_address);

   }
}
```

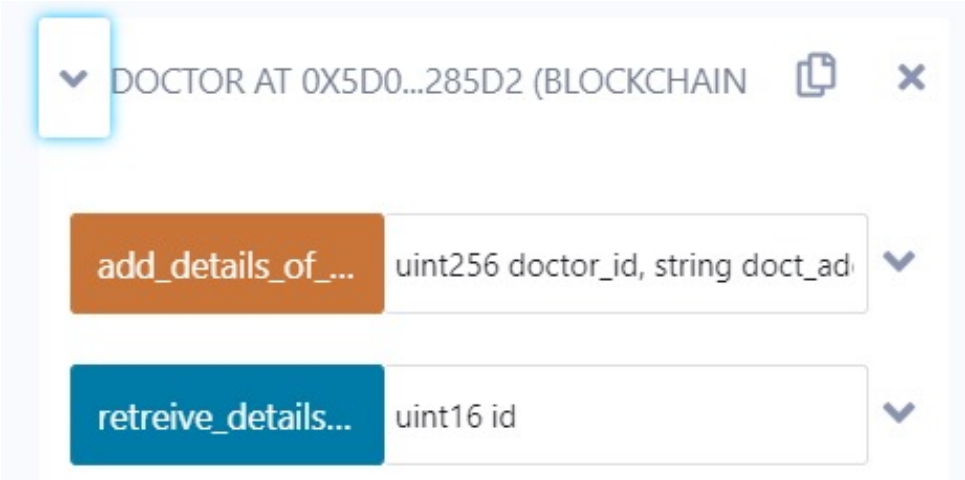
Deploying Doctor.sol using metamask account



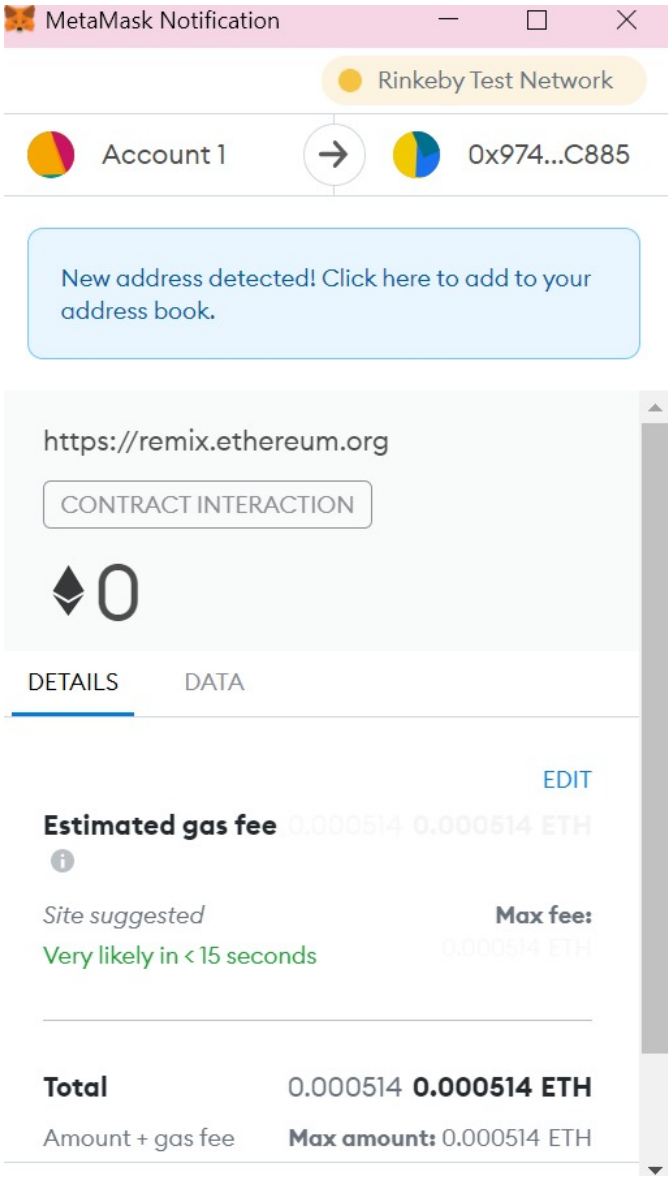
Transaction Got Confirmed



Doctor Node Successfully Created



Calling add_details_of_doctor function using metamask account



Displaying the updated details of doctor using retreive_details_of_doctor function