Final Paper

- **Project Title: Secure digital identity and vaccine passport management**
- **Team Members: Zihao Jing, Wenliya Lyu, Shulan Gan, Chang Liu**
- **Date : 8.26.2022**
- **Email Address : 1971008968@qq.com**

Abstract

Vaccination records are currently regulated differently in different regions, which is inconvenient for people traveling across regions. A consensus vaccination record book needs to be created. Blockchain has consensus-building mechanisms that use hashes to ensure that the data that has been written is not altered, which can ensure that vaccination records that have been written are tamper-proof. So that consensus is formed.
To implement the idea, a Dapp called V-Pass was created in which users bind an Ethereum account to the app and bind it with their real passport to ensure that the record of the current account is theirs. The user uses the app account to ask the hospital to add the vaccination record for them when they get vaccinated, and displays the relevant interface of the record information in the app when they need to show the vaccination record.

The application uses the **front-end framework vue3** to create the client, uses **solidity** to write smart contracts, uses **javascript** and **typescript** to create interactive programs, uses **metamask** as a relay to publish transactions, and uses **axios** to send **JSON-RPC API** to Ethereum. Sending requests uses interfaces such as eth_call to interact with the contract.The front-end part of the application has been deployed to the relevant **cloud server**, so that we can access and use the application through the URL. The contract part is deployed to Ethereum's **Reposten test network** using the **remix IDE**.

The application will likely be favored by WHO and other health organizations because of its simplicity and ease of use and high reliability. Once the vaccine information is recorded, it cannot be modified. It will be widely promoted and used to solve the problem of vaccination records in various regions. The problem of not being recognized by each other.

1. Introduction

1.1 Background

All lives have changed as a result of the current COVID-19 pandemic epidemic dramatically, and civilization underwent new stages with new provisions. This article details a solution of a system of digital certificates for the secure storing of one of these demands. Our project developed a system that is able to record people’s vaccination history and is able to help institutes and organizations to tell it the people is qualified to entrance the countries or regions. The system being proposed is based on the advantages of blockchain technology, as well as creating a decentralized, mobile, and cross-platform application for its use, as well as a system of incentives and a personalized cryptocurrency to reward users who use it.

1.2 Project Idea

The introduction of the vaccine passport can assist persons who have received vaccinations in returning to regular life as quickly as feasible. The concept behind a vaccine passport, or a “digital health identity” is digital certificate to prove vaccination identify vaccinated individuals give access to certain public spaces or transportation. In this project, we use the technology of blockchain to guarantee the security of the personal data. The adoption of blockchain technology must, however, ensure that people's personal information, particularly their medical information, can be kept secret owing to invasions of privacy. It should just check that people are immunized and not disclose or keep any personal information. There are mainly three objectives we aims to realize in this project:

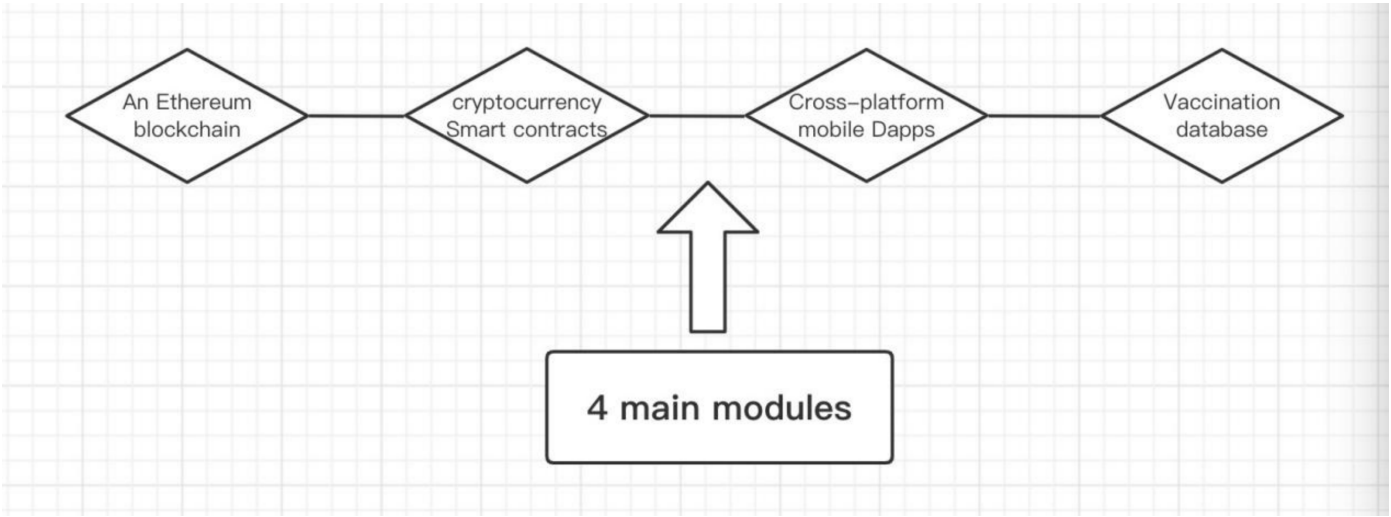
Firstly, we aims at reducing privacy invasion and prevent discrimination against those non-vaccinated.

Secondly,we are to ensure the security of vaccination passports and prevent tampering and counterfeiting.Thirdly, our vaccine passport update digital identity timely and gain regional recognition.Last but not least, our system are designed to be easy to use for everybody and the information safety is also guaranteed.

2. Design Patterns

This section contains hardware and software support for the various modules of the app, explaining the app's role in the user's vaccination. And draw the UML diagram, activity diagram and other necessary elements of the program design, better understand the design pattern of the v-pass application, and make a general overview for the next part to explain the development of the application.

2.1 Main modules



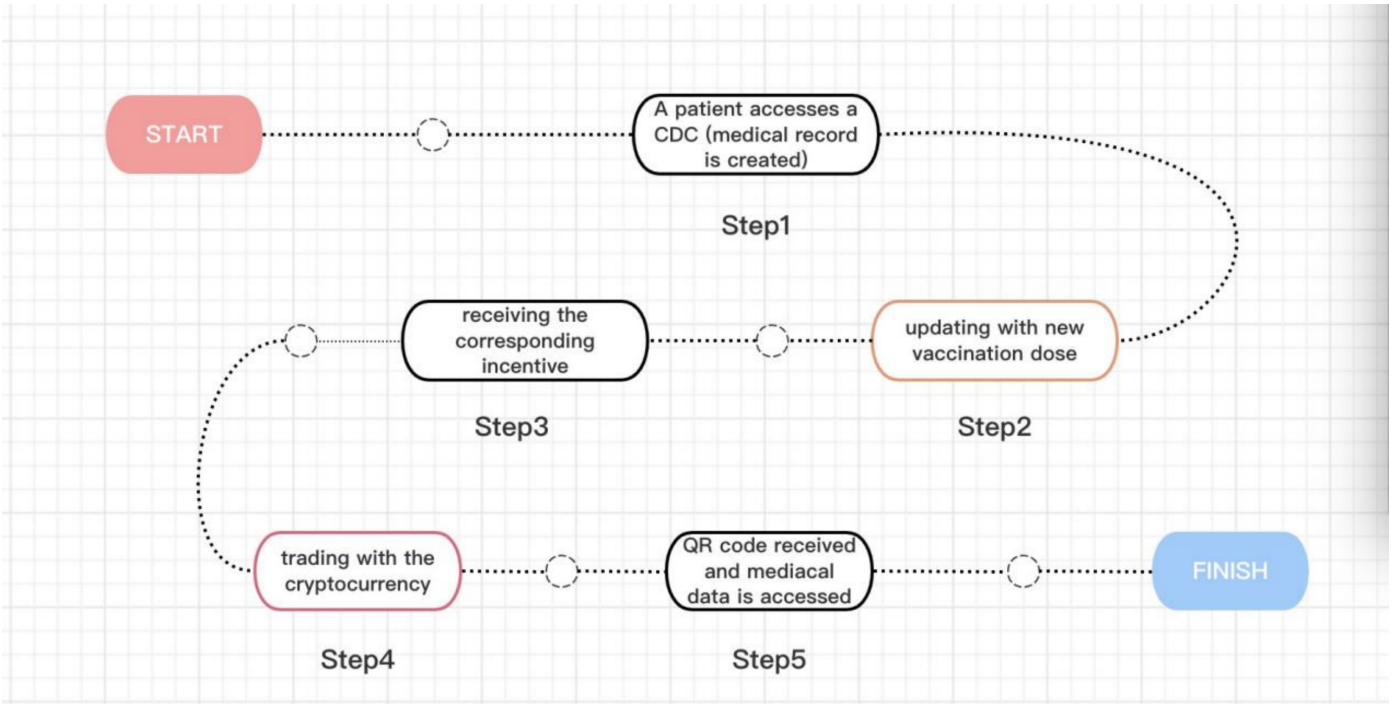
• An Ethereum blockchain with several mining nodes that accept transactions.

• cryptocurrency Smart contracts. These were developed in Solidity with the Remix online IDE.
implement the medical record contract, write and update data,which enables an incentive system that also allows trading. The Faucet smart contract is created to supply the demand of Ether to the different users of the system (hospital, CDC, individuals)

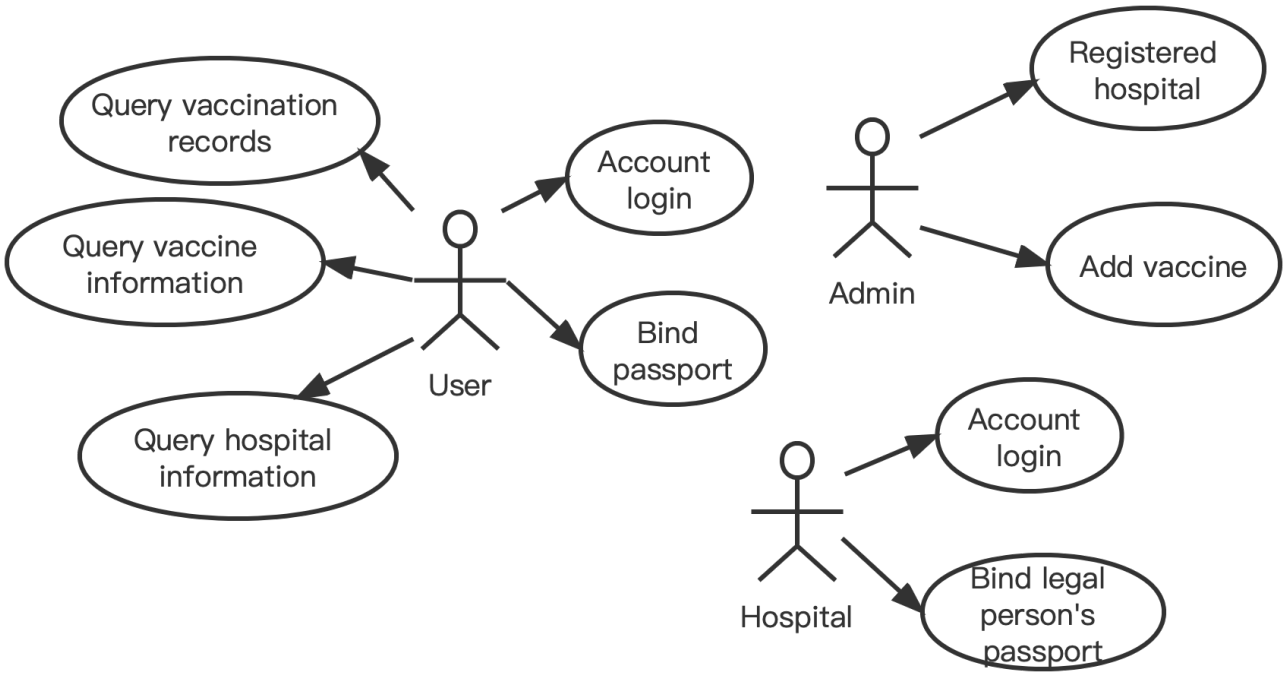
• Cross-platform mobile Dapps developed to modularize the project and divide the functionality on the Blockchain: reading for patients and writing for CDCs

• Vaccination database.

2.2 Usage Process

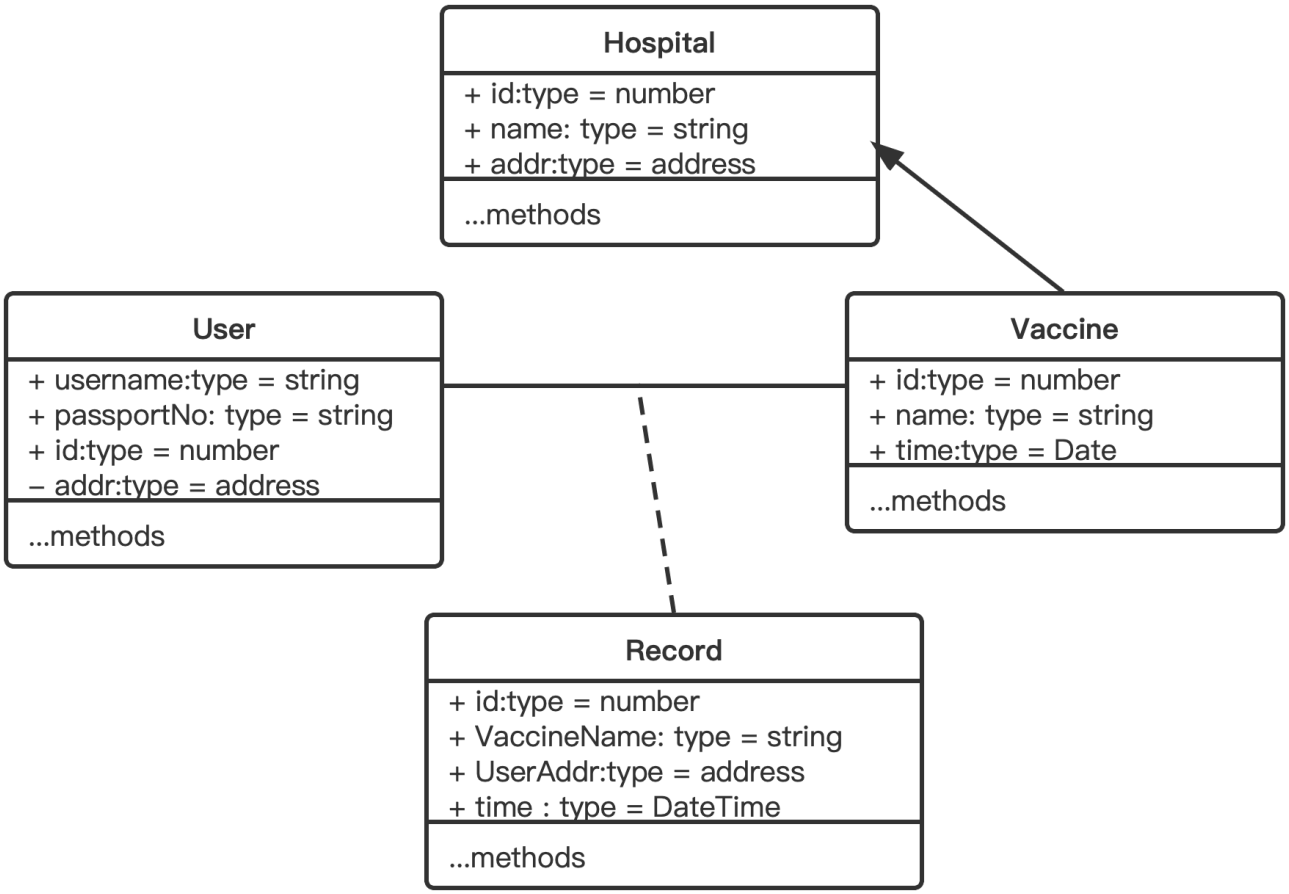


2.3 Use Case Diagram



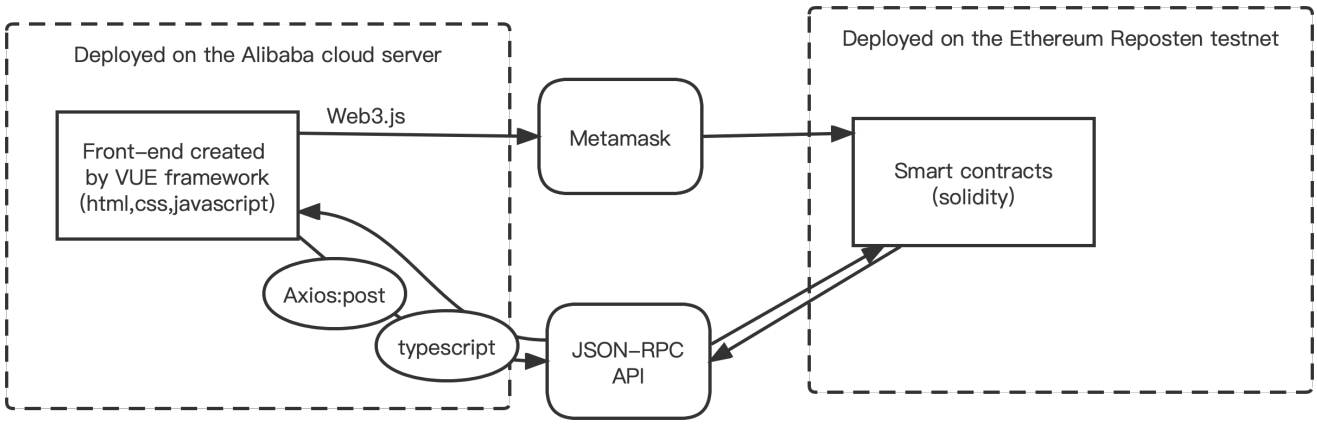
This application has three main users, patients, administrators and hospitals. There are different client portals for each user, with different functions and permissions. Effectively ensure data security and software stability.

2.4 Class Diagram



Using object-oriented thinking to look at the application to be implemented, there are mainly four classes, the class diagram is as above.

2.5 Technology stack



The above picture shows the main working principle of the application, which is a complete application from client to server, with complete design and function implementation.

3. DApp Development

Let's take a look at the specific development details of the application.

3.1 Application Development-Front end

3.1.1 Import modules

main.ts Introduces naive-UI components to help rapid prototyping of our applications.

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import naive from 'naive-ui'

createApp(App).use(router).use(naive).mount('#app')
```

Import of other important modules...

```
import Web3 from 'web3'
import {ref} from 'vue'
import { Icon } from '@vicons/utils'
import { Copy16Regular } from '@vicons/fluent'
import axios from 'axios'
import requests from '../request'
import { sleep } from 'seemly'
import {onMounted} from 'vue'
import utils from '@/Utils'
import { useRouter } from 'vue-router'
import { now } from 'lodash'
```

The file tree as follow...


MY-DAPP
contracts
passport_v3.2.sol U
passport.sol U
dist
node_modules
public
src
assets
components
router
index.ts M
type
type.d.ts U
views
AboutView.vue
HomeView.vue
HospitalPage.vue U
PassportMain.vue U
UserLogin.vue U
UserPage.vue U
App.vue M
main.ts M
request.ts U
shims-vue.d.ts
Utils.ts U
.browserslistrc
.eslintrc.js
.gitignore
babel.config.js
package-lock.json M

3.1.2 Main functions of user client

When a user logs in, a metamask client needs to be connected to connect to the user's Ethereum account. After the connection is successful, save the relevant information and go to the next step.

```
function signAccount(){
    if(window.ethereum){
        window.ethereum.enable().then(res=>{
            myAccount.value = res[0];
            utils.setCookie('account',myAccount.value)
            utils.setCookie('contract',contractAddr)
            showModalRef.value=true;
        })
    }
    else{
        alert('Please install metaMask!')
    }
}
```

In order to be able to uniquely bind the user's Ethereum account address to the user, the user needs to operate and bind their real passport. To confirm identity when performing operations such as vaccination records. Binding a passport requires writing data to the blockchain, so use window.ethereum.request to call metamask to initiate a transaction.

```
async function bindPassport(){
    let data = await requests.getFunc(functions[0])
    data = data +
'0000000000000000000000000000000000000000000000000000000000000060000000000000000000000000
00000000000000000000000000000000000000a0'
    data = requests.addAddress(data,myAccount.value.slice(2))
    data = requests.addParamsString(data,name.value)
    data = requests.addParamsString(data,passportNo.value)
    console.log(data)
    const transactionHash = await window.ethereum.request({
        method:'eth_sendTransaction',
        params:[
            {
                'to' : contractAddr,
                'from': myAccount.value,
                'data': data,

                "gasPrice": "0x184e72a000",

            }
        ],
    });
    console.log(transactionHash)
}
```

When the user client interface is rendered, the current account address and global contract address saved during login are obtained from the cookie：

```
onMounted(()=>{
    AccountShow.value = utils.getCookie('account')
    myAccount = utils.getCookie('account')
    contractAddr = utils.getCookie('contract')

    initialize_VaccineList();
    initialize_HospitalList();
    initialize_Record();
})
```

The user client has the function of querying its own vaccination records, querying all vaccine information, and hospital information. So the data corresponds to three lists.

The following is the initialization operation of the three lists...

```
async function initialize_VaccineList() {
    let data = await requests.getFunc(functions[0])
    let re = await requests.eth_call(contractAddr,data)
    const len = requests.hex2dec(re)
    let array = [];
    for(let i=0;i<len;i++){
        data = await requests.getFunc(functions[1])
        data = requests.addParamsNum(data,i)
        re = await requests.eth_call(contractAddr,data)
        let name = requests.str_decode(re,3)
        let id = requests.str_decode(re,5)
        let time = requests.str_decode(re,7)
        array.push({name:name,id:id , time:time})
    }
    VaccineList.value = array;
}
```

```
async function initialize_HospitalList() {
    let data = await requests.getFunc(functions[2])
    let re = await requests.eth_call(contractAddr,data)
    const len = requests.hex2dec(re)
    let array = [];
    for(let i=0;i<len;i++){
        data = await requests.getFunc(functions[3])
        data = requests.addParamsNum(data,i)
        re = await requests.eth_call(contractAddr,data)
        console.log(re)
        let name = requests.str_decode(re,3)
        let id = requests.hex2dec(re.slice(0,66)).toString()
        let  addr = requests.addr_decode(re,2)
```

```
        array.push({name:name,id:id,address:addr})
    }
    HospitalList.value = array;
}
```

```
async function initialize_Record(){
    let data = await requests.getFunc(functions[5])
    data = requests.addAddress(data,myAccount.slice(2))
    let re = await requests.eth_call(contractAddr,data)
    const len = requests.hex2dec(re)
    let array = [];
    for(let i=0;i<len;i++){
        data = await requests.getFunc(functions[4])
        data = requests.addAddress(data,myAccount.slice(2))
        data = requests.addParamsNum(data,i)
        re = await requests.eth_call(contractAddr,data)
        console.log(re)
        let name = requests.str_decode(re,4)
        let id = requests.str_decode(re,6)
        let time = requests.str_decode(re,8)
        array.push({name:name,id:id , time:time})
    }
    RecordList.value = array;
}
```

3.1.3 Main functions of hospital client

The hospital client has three main functions, namely hospital registration, adding vaccine information for users, and adding new vaccines.

```
async function addHospital(name:string,addr:string){
    let data = await requests.getFunc('createHospital(string,address)')
    data += '0000000000000000000000000000000000000000000000000000000000000040'
    data = requests.addAddress(data,addr.slice(2))
    data = requests.addParamsString(data,name)
    const transactionHash = await window.ethereum.request({
        method:'eth_sendTransaction',
        params:[
            {
                'to' : contractAddr,
                'from': myAccount.value,
                'data': data,

                "gasPrice": "0x84e72a000",

            }
        ],
    });
```

```
}
```

```
async function addRecord(user:string,name:string,id:string){
    let data = await requests.getFunc('addVaccine(address,string,string,string)')
    data = requests.addAddress(data,user.slice(2));
    data += '0000000000000000000000000000000000000000000000000000000000000080'
    data += '00000000000000000000000000000000000000000000000000000000000000c0'
    data += '0000000000000000000000000000000000000000000000000000000000000100'
    data = requests.addParamsString(data,name)
    let nowDate = new Date();
    let strDate = nowDate.getMonth()+'/' +nowDate.getDate()+'/'+ nowDate.getFullYear()
    data = requests.addParamsString(data,strDate)
    data = requests.addParamsString(data,id)
    const transactionHash = await window.ethereum.request({
        method:'eth_sendTransaction',
        params:[
            {
                'to' : contractAddr,
                'from': myAccount.value,
                'data': data,
                "gasPrice": "0x84e72a000",
            }
        ],
    });
}
```

```
async function addVaccineRaw(name:string,id:string){
    let data = await requests.getFunc('addVaccineRaw(string,string,string)')
    data += '0000000000000000000000000000000000000000000000000000000000000060'
    data += '00000000000000000000000000000000000000000000000000000000000000a0'
    data += '00000000000000000000000000000000000000000000000000000000000000e0'
    data = requests.addParamsString(data,name)
    data = requests.addParamsString(data,id)
    let nowDate = new Date();
    let strDate = nowDate.getMonth()+'/' +nowDate.getDate()+'/'+ nowDate.getFullYear()
    data = requests.addParamsString(data,strDate)
    const transactionHash = await window.ethereum.request({
        method:'eth_sendTransaction',
        params:[
            {
                'to' : contractAddr,
                'from': myAccount.value,
                'data': data,
                "gasPrice": "0x84e72a000",
            }
        ],
    });
}
```

The three main functions of the hospital client all need to write data into the blockchain, so they all need to call metamask to initiate a transaction, and the initiator bears the corresponding gas fee.

3.2 Application Development - Interact

Web3js is a very useful tool for interacting with smart contracts. However, in this application development, after many attempts and explorations, it was impossible to successfully install the application web3js and use the packaged functions. So I chose JSON-RPC API for interacting with smart contracts. In the interaction, we also encountered and solved many problems. For example, there is no clear and clear guidance for the parsing and coding of the data field in the API, and the intelligence can write the correct function through its own exploration and regular search.

In order to facilitate the various modules of the application to call related interactive functions, I encapsulate the interactive functions and some auxiliary functions in a .ts file: requests.ts, some of the functions as follow :

```
async eth_call (to:string,data:string) {
        const params = [{
            "to": to,
            "data":data
        },"latest"]
        frameDs['params']=params
        frameDs['method']="eth_call"
        console.log(frameDs)
        let re = ''
        await axios.post(postAddress,frameDs).then(res=>{
            console.log(res.data)
            re = res.data.result
            //console.log(re)

        })
        return re
    },
    async eth_accounts(){
        frameDs['params']=[];
        frameDs['method']='eth_accounts'
        await axios.post(postAddress,frameDs).then(res=>{
            console.log(res.data)
        })

    },
    stringToHex(str:string){
            let val="";
                for(let i = 0; i < str.length; i++){
                if(val == "")
                        val = str.charCodeAt(i).toString(16);
                    else
                        val += str.charCodeAt(i).toString(16);
```

```
        }
        return val;
    },
    async str_sha3(str:string){
        const restr = '0x'+this.stringToHex(str);

        const params = [restr];
        frameDs['params']=params
        frameDs['method']="web3_sha3"
        console.log(frameDs)
        let result = '';
        await axios.post(postAddress,frameDs).then(res=>{
             result = res.data.result
             //console.log('re'+result)
        })
        return result;
    },
    async getFunc(str:string){
        let data = await this.str_sha3(str)
        data = data.slice(0,10)
        return data;
    },
    addParamsNum(str:string,num:number){
        const num_temp = num.toString(16);
        let num_re = ''
        if(num_temp.length<64){
            for(let i=0;i<64-num_temp.length;i++)
            {
                num_re+='0'
            }
        }
        num_re+=num_temp;
        return str+num_re;
    },
    addAddress(str:string,addr:string){
        let add_re = ''
        if(addr.length<64){
            for(let i=0;i<64-addr.length;i++)
            {
                add_re+='0'
            }
        }
        add_re+=addr;
        return str+add_re;
    },
    addParamsString(str:string,s:string){
        const len = s.length;
        const len_str = this.addParamsNum('',len)
        let s_temp = this.stringToHex(s)
```

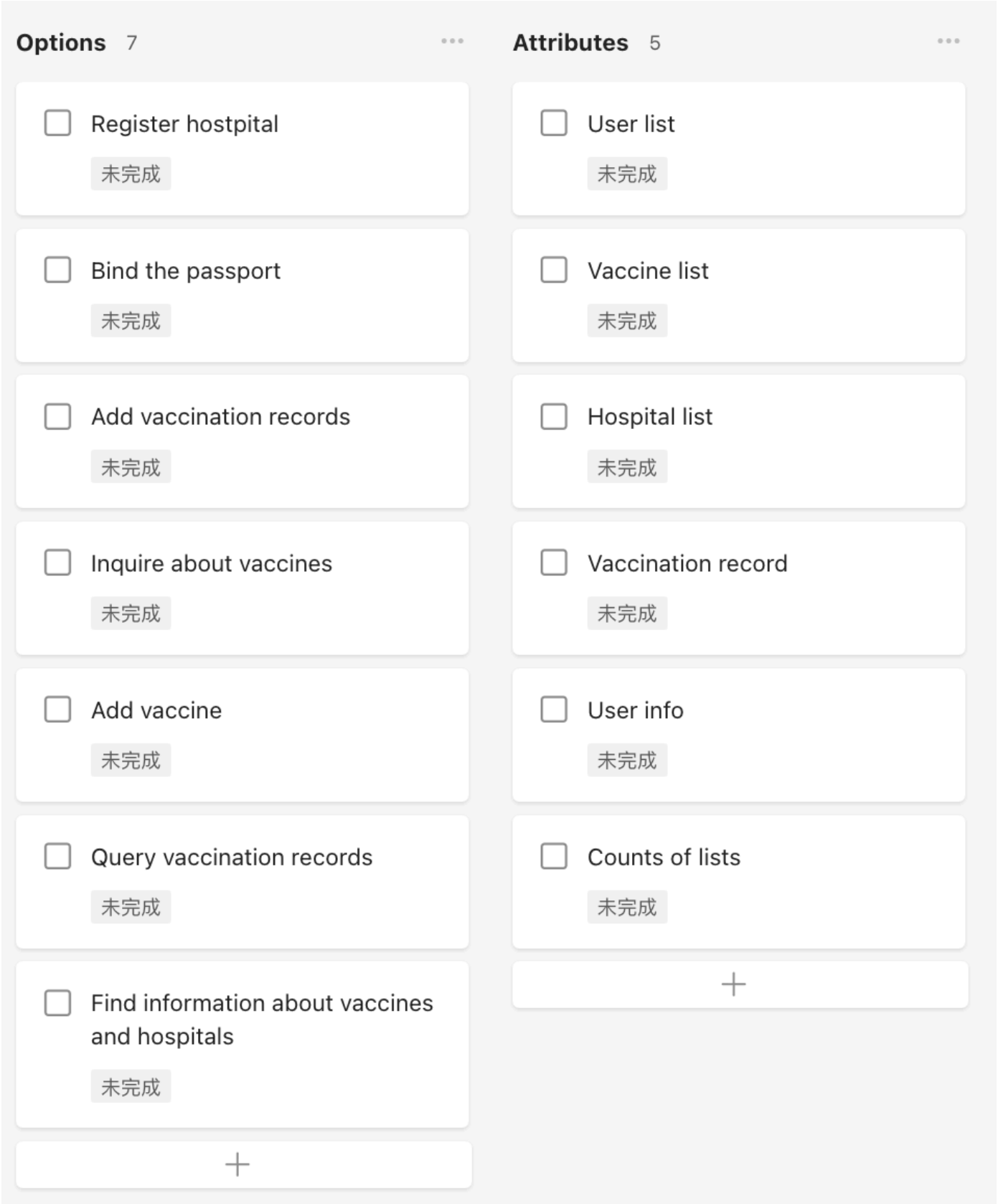
```
        for(let i=0;i<64-len*2%64;i++)
        {
            s_temp+='0'
        }

        return str+len_str+s_temp;
    },
     str_decode(restr:string,start:number){
        const str = restr.slice(2)
        const a=start*64;
        const len = str.slice(a,a+64)
        const leng = parseInt(len,16)
        const ori_code = str.slice(a+64,a+128)
        let val=''
        for(let i = 0; i < leng; i++){
            val += String.fromCharCode(parseInt(ori_code.substr(i*2,2),16));
        }
        return val;
    },
    addr_decode(restr:string,start:number){
        const str = restr.slice(2)
        const a=start*64;
        const ori_code = str.slice(a,a+64)
        console.log(ori_code)
        const val = ori_code.slice(24,64)
        return '0x'+val;
    }
```

Each module in the application implements the encoding and decoding of parameters by calling these functions to ensure correct interaction with the smart contract.

3.3 Application Development - Smart Contract

In the design of smart contracts, I use the teambition tool to record the application's requirements for functions and functions, so that our design and writing can be more coherent and logical.



3.3.1 Storage Data

There are mainly three parts of structs using for storing data, which are Vaccine, User, Hospital, including each of the information in it.

```
    struct Vaccine {
        string name;
```

```
        string id;
        string time;
    }
    // User basic information
    struct User {
        uint256 id;
        string name;
        string passportNo;
        address addr;
    }
    // Hospital basic information
    struct Hospital {
        uint256 id;
        string name;
        address addr;
    }
```

In addition, mapping is used for matching data by its address.

```
    // match hospital address with information
    mapping (address => Hospital) hospital;
    // match user address with information
    mapping (address => User) user;
    // match user address with its total vaccine count
    mapping (address => uint32) vaccineCount;
    // match user address with its vaccine
    mapping (address => mapping(uint32 => Vaccine)) vaccine;
```

3.3.2 Modifiers

Secure and privacy are important when accessing data in the smart contract. Due to that, various modifiers are used in contract to limit users' data accessing (only allow them access their own data). In addition, owner, the manager, has top root in accessing and modifying.

```
  modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    modifier onlyHospital() {
        require(
            msg.sender == hospital[msg.sender].addr ||
            msg.sender == owner
            );
        _;
    }
    modifier onlyUser() {
        require(
            msg.sender == hospital[msg.sender].addr ||
```

```
            msg.sender == owner ||
            msg.sender == user[msg.sender].addr
            );
        _;
    }
```

3.3.3 Update data functions

After initializing and deploying the smart contract, the manager needs to add authorized hospitals, having right to provide public vaccine which also should to be inserted in the contract.

```
 function createHospital(string memory _name, address _hospital) public onlyHospital{
        require(hospital[_hospital].addr != _hospital);
        Hospital memory h = Hospital(
            hospitalCount,
            _name,
            _hospital
        );
        hospital[_hospital] = h;
        hospitalArray.push(h);
        hospitalCount++;
    }
    function addVaccineRaw(string memory _name,string memory _id,string memory _time)
public onlyHospital {
        Vaccine memory newVaccine = Vaccine(_name,_id,_time);
        vaccines.push(newVaccine);
        vaccineCountAll ++;
    }
```

Those authorized hospitals then have chances to add users' information and their vaccine into the contract by the following functions.

```
 function createUser(string memory _name,string memory _passportNo ,address _user)
public onlyHospital {
        require(user[_user].addr != _user);
        User memory u = User(
            userCount,
            _name,
            _passportNo,
            _user
            );
        Vaccine memory v = Vaccine("", "", "");
        user[_user] = u;
        vaccineCount[_user] = 0;
        vaccine[_user][vaccineCount[_user]] = v;
        userCount++;
    }
```

```
    function addVaccine(address _user, string memory _name,string memory _time, string
memory _id) public onlyHospital {
        require(_user == user[_user].addr);
        Vaccine memory v = Vaccine(
            _name,
            _id,
            _time
        );
        vaccine[_user][vaccineCount[_user]] = v;
        vaccineCount[_user] ++;
    }
```

During the creating, the methods use those modifiers in the former part to ensure that no one can insert data without limitation and supervision.

3.3.4 Get data functions

In these methods, modifiers are also used and a free access way is chosen when accessing data, which means that there is no cost when users call functions and get data.

```
  // Get user's information
    function getUserInfo(address _user) public onlyUser view returns(User memory
userinfo) {
        userinfo = user[_user];
    }

    // Get user's vaccine information with certain index
    function getUserVacc(address _user, uint32 _index) public onlyUser view
returns(Vaccine memory v) {
        uint32 count = vaccineCount[_user];
        require(count > 0);
        require(_index < count);
        v = vaccine[_user][_index];
    }

    // Get user's total vaccine number
    function getUserVaccCount(address _user) public onlyUser view returns(uint32 count)
{
        count = vaccineCount[_user];
    }

    // Get hospital information
    function getHospital(uint32 _index) public onlyUser view returns(Hospital memory v)
{
        v = hospitalArray[_index];
    }
```

Now, we have written all the functions of the application and implemented the expected function. Now it's time for us to test and deploy.

3.4 Application Development - Software test

The software testing phase of the application is performed locally. Execute `npm run serve` in the root directory of the front-end file. The application will be run on `localhost:8080`. Fortunately, after many days and nights of tireless work, the features of the app passed the test and all the issues found were fixed, which makes the application ready for deployment.

3.5 Application Development - Deployment

- Front-end

  Run command `npm run build` in root directory in vue project, the a directory dist will be created. Then the dist derectory was send to cloud server. In the Aliyun domain name management system, a second-level domain name is parsed for the application, and **nginx** is used to configure and run the dist file. Then we can access the application with url.

  The directory is renamed as vpass.

```
server {
        listen 80;
        server_name vpass.selmiss.xyz;
        location / {
          root  /home/selmiss/vpass;
          index index.html index.htm;
          try_files $uri $uri/ /index.html;

        }
        location @router {
                rewrite ^.*$ /index.html last;
        }
}
```

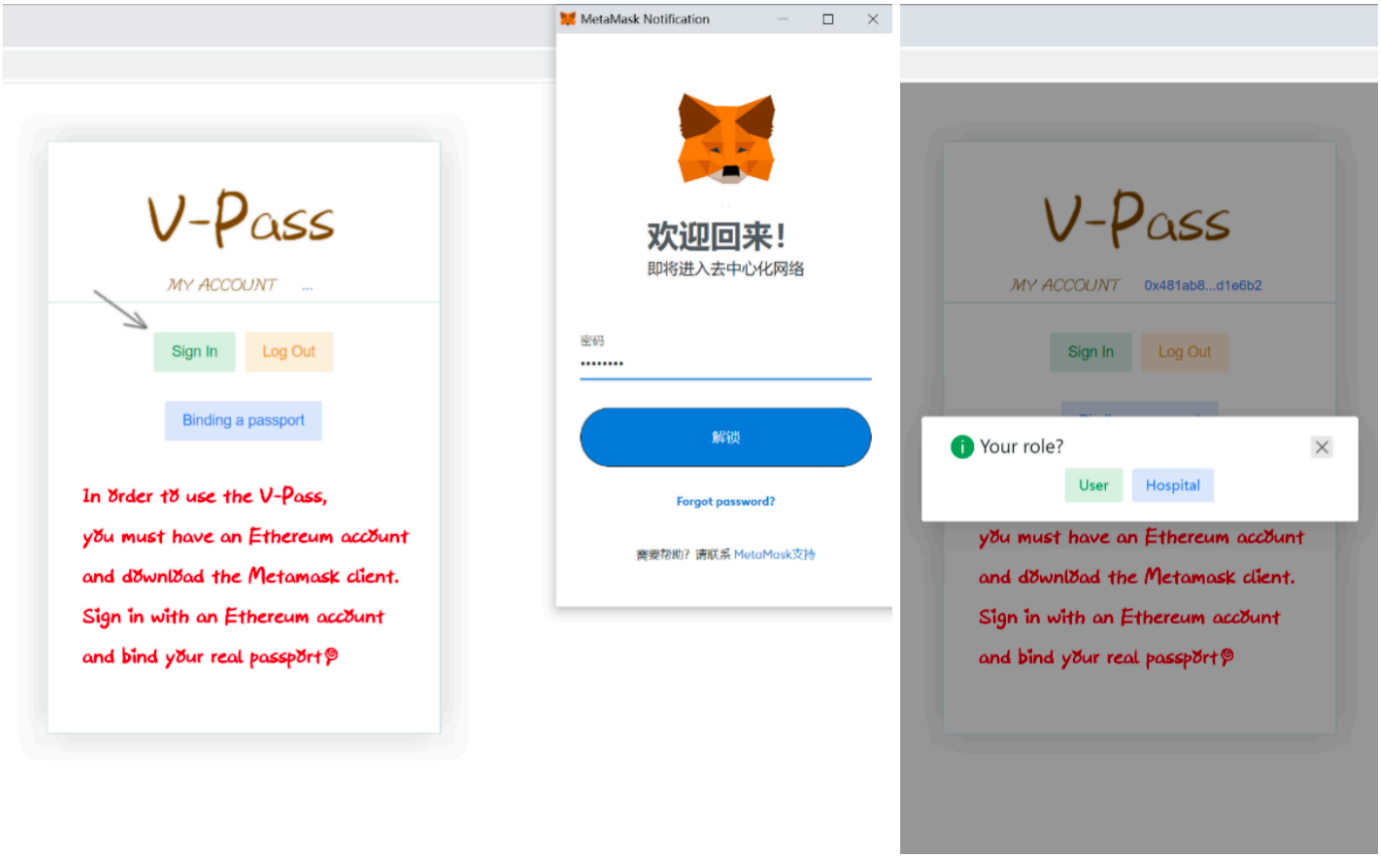
- Smart contract

  It is very convenient and quick to use remix IDE for debugging and deployment when deploying smart contracts. This is deployed in the Ethereum reposten test network

4. Application Usage

In the user interface, it provides two different parts for different kinds of users including public and CDCs. In addition, learnability needs to be considered, which means new users can begin effective interaction and achieve maximal performance.

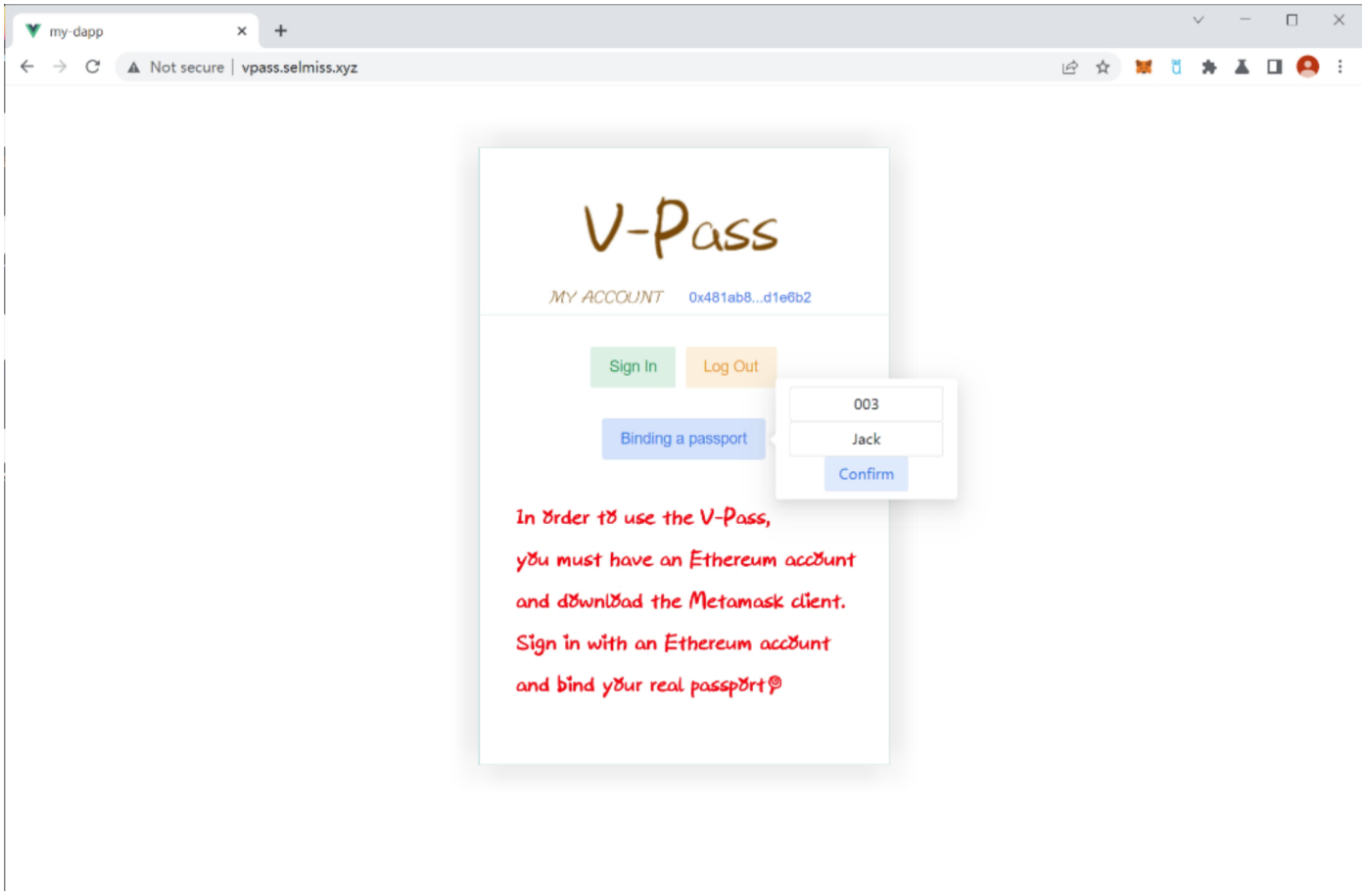
4.1 Login



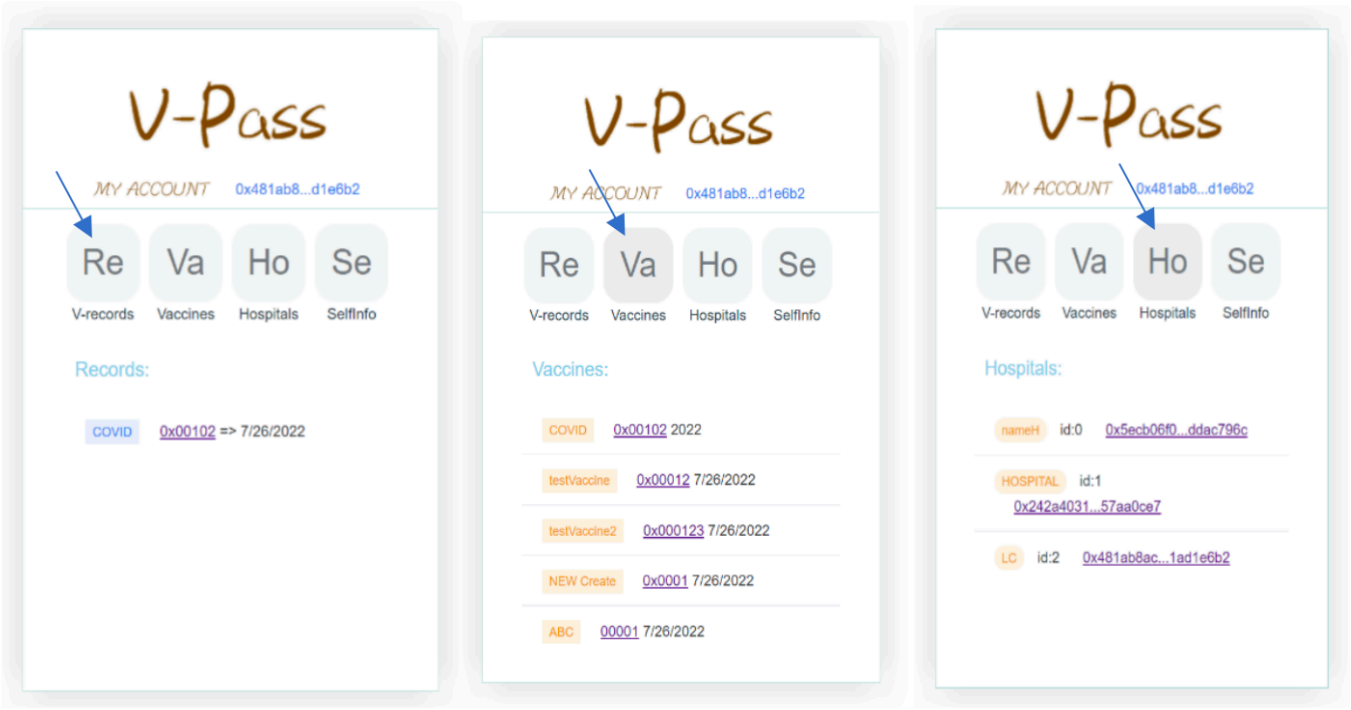
After login, users can choose their role and access to the smart contract.

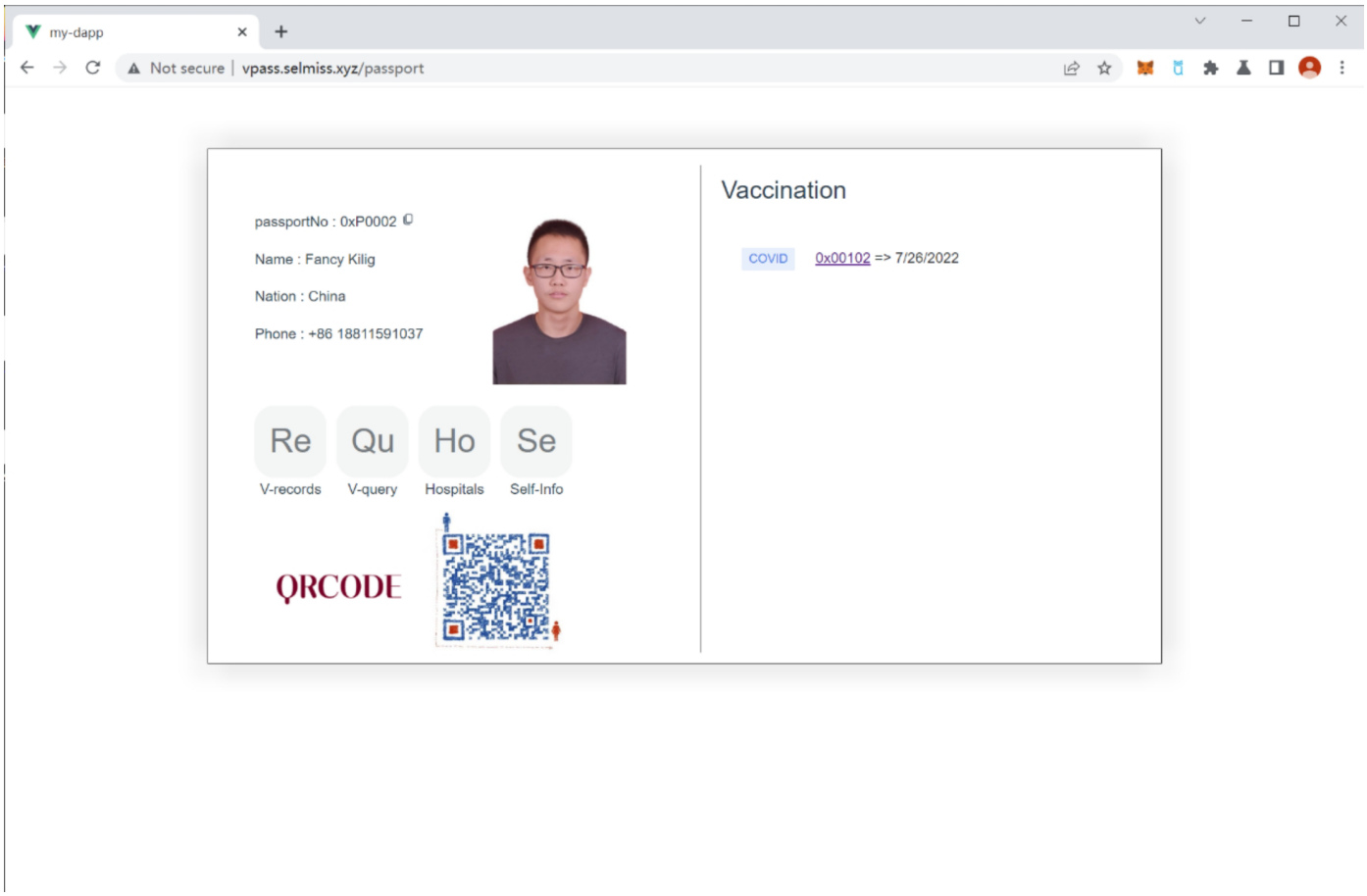
4.2 User Client

At the beginning, users should bind their passport to the contract.



In the user client, there are mainly for panels to show data including vaccine records, vaccines, hospitals and users' own information.

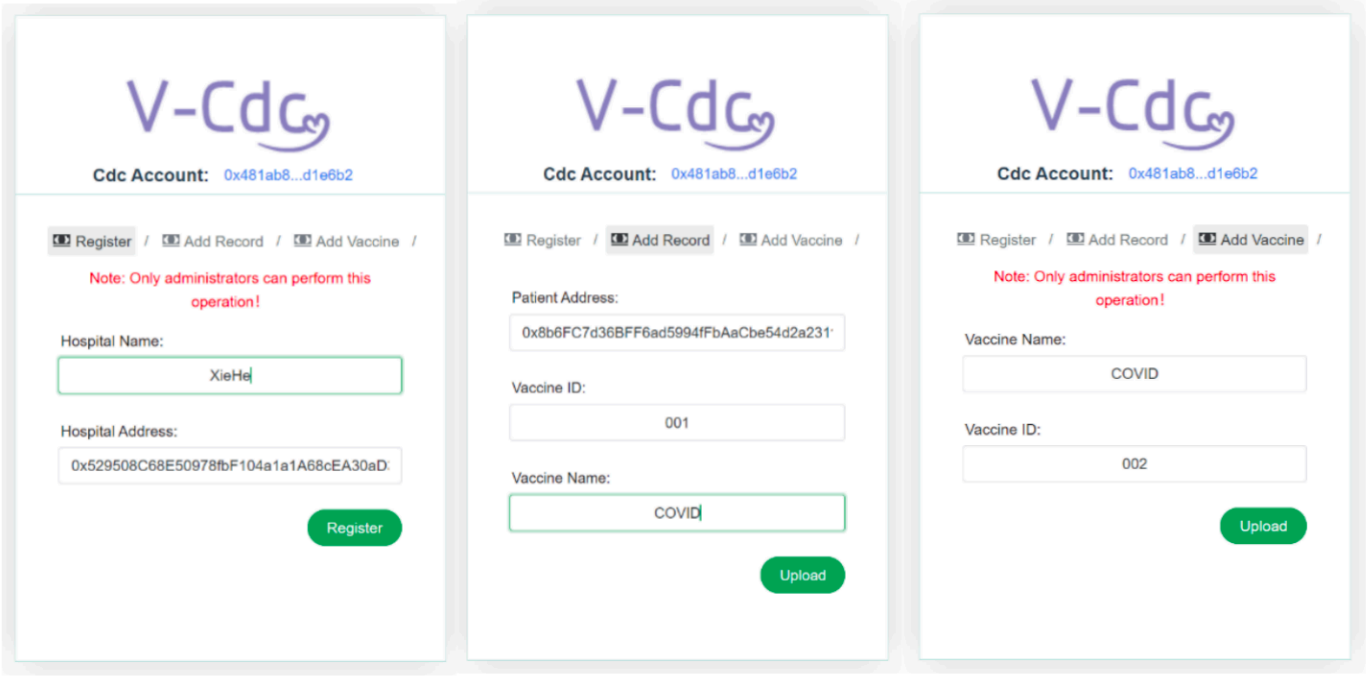




In addition, a QR code can show information easily and when scanning it, data will be sent to receivers.

4.3 Csc Client

In the CDCs client, it is mainly used for inserting vaccine records and vaccine and hospital information.



There are some accessing limitations in this client, with only administrator or manager can add authorized hospital and vaccine information.

5. Future of V-pass

The goal was to develop a robust system that addresses several pressing COVID-19 pandemic issues from a technological (as a new COVID-19 information storage system), medical (favoring the development of low health risk circumstances), and social standpoint (waiving restrictions as well as economically benefiting both customers and companies).

However, there are several progresses we aim to realize in the future. First, the vaccination history can be presented in the form of QR code with color(green represents qualified vaccination, red ones vice versa).Secondly, this project can also be applied to the nuclear acid test, while one was tested, the record is added to the blockchain. Furthermore, this project has the vision and possibility to be applied to a wider medical use, such as a electronic medical record which is able to record patient's past medical history. Based on the electronic medical record, qualified institutes and organizations can call the records of patient when necessary. For instance, a people intends to participate in bungee jumping, the amusement park staffs are able to call his or hers medical history to check if he or she is able to attend in this activity and assess the risk for them. There may be more functions and applications that we can extend this project and make a change for this world, we are all working and looking forward.

6. Solution of Hard Problems

6.1 Promotion of V-pass

6.1.1Differences of V-pass from Others

Ordinary vaccinations often use paper vaccination books, or electronic vaccination certificates that only a small area such as a city or county can form a consensus on. During the Covid-19 outbreak, it can be seen from the epidemic prevention and control applications across China that it is difficult to have a nationwide health application. Vaccinations vaccinated in one city are often difficult to find through the app in another city. The V-pass application adopts the decentralized technology of blockchain Ethereum, which enables authorized hospitals to add vaccination records and write them into the blockchain database, thereby ensuring that the added vaccination records are not modified. Such an application can ensure that more cities reach a consensus and increase the recognition of the application.

6.1.2 Get Authoritative Recognition

It is very important to get the recognition of the government, health organizations and other institutions, which can accelerate the promotion of the application. At the same time, because of some features of our application, it is easier to be accepted by authoritative organizations. Through authoritative recognition, the promotion of the application will be greatly improved. With a higher promotion, more organizations will recognize it. This is a positive cycle.

6.2 Who will pay the gas fee?

6.2.1 User binding

Users do not need to pay gas fees when logging in with an Ethereum account. When you need to start using the v-pass application, you need to use a real passport to bind your identity, and the user should pay the gas fee. Users do not need to initiate transactions when using other functions, so no fees are required.

6.2.2 Add hospitals and vaccines

Adding hospitals and vaccines needs to be operated by the application administrator, so the application has a small-scale operation and maintenance team to connect with various regions to ensure the correctness of adding vaccines and opening hospital permissions. At this point, the operation and maintenance team initiates the transaction and pays the gas fee.

6.2.3 Add vaccination records for users

This operation is performed by the hospital, which initiates the transaction and pays the gas fee for adding the record. Of course, the hospital can charge the user for this part of the fee when serving the user.

6.3 Permissions issue

6.3.1 User

Users have permissions to log in, bind passports, and query.

6.3.2 Admin

The administrator is the operation and maintenance team of the application, and their responsibility is to ensure the legitimacy and authority of registered hospitals and the authenticity of vaccine information. Therefore, adding new vaccines and registering hospitals will be initiated by the hospital and executed after the operation and maintenance team has reviewed and approved it. This ensures the authenticity of the information queried by the user.

6.3.3 Hospital

The hospital can add vaccination records for users, but the premise is that the hospital needs to register, bind the passport of the legal person and obtain the approval of the operation and maintenance team.

6.4 Private issues

The user's privacy issue is also crucial, which is mainly reflected in this application that other people and institutions cannot query the user's vaccination records. That is to say, only the user himself can query the vaccination record and show it to others for inspection, which ensures the privacy of the user. These are guaranteed by setting the calling permissions of individual functions.

7. Summary

7.1 Our Work

In this project, under the guidance of the professor, we improved our ideas step by step, and finally formed a mature idea. We use what we learn in the course to create a decentralized app. However, this process is still full of hardships, such as how to use the PRC interface to interact with the contract, how to correctly initiate transactions, and how to make the logic self-consistent. Many times, we work until four in the morning in order to be able to successfully create the app.

Fortunately, before the deadline, we solved most of the problems, successfully created the v-pass application, and deployed it on the server.

7.2 Project Harvest

In this course study and project development, I learned a lot of theoretical knowledge of the blockchain Ethereum. And through practical learning in the development of a Dapp, I have mastered the whole process of Ethereum decentralized app development, and can use its services in our daily life. Although we have paid a lot of hard work, we have learned a lot and gained a lot, which is very gratifying.