1 Target Audience

1.1 Huge Legacy

We target people who has huge legacy, that the amount is big enough for them to be worried while they are alive. Or even be cautioned by others that they should do something about it.

1.2 Complex Inherit Rules

The target could have complex inherit rules, including trust, complex percentage or even company board seats.

1.3 Complex Inherit Relationship

A lot of people might pop up and claim their rights when our TA passes away. Our TA could have a complex inherit relationship that often causes law suits or arguments.

1.4 High Time Cost

Our TA are set to be busy and might have high time cost. If they spend a lot of time dealing with these, kinds of things will be costly.

1.5 Privacy of Testament

There might be some requests that our TA don't want to reveal their content of testament. Revealing their testament might cause unnecessary problems.

2 Pain Point

2.1 Tampering

In our current law system, there are still a lot of rooms for malicious tampering. There are always risks that others can access the content of a testament. For example, if any testament is in digital form, the fire wall can never really guarantee it won't be accessed.

2.2 Time-consuming

Because of our TA's feature, it will cause big problems if the testament is distorted by others. The existing system that prevents these kinds of situations are highly time consuming. In fact, the more complex the method is, the more difficult others can distort the content.

2.3 Problems about Complex Rules

If anyone wants to set a complex rule of testament, it's gonna take more time than a normal testament. And how to enforce the rules in the future is also a big problem. If others do not follow, it's going to be a lot of endless law suit.

3 Solution

3.1 Pain point 1: Tampering versus Untamperable on Mechanism Level

Block Chain technology is based on a data structure built up by hash

pointers and hash pointers can keep the all information on the same blockchain related tightly to each other. That means if you want to make any piece of information tampered and maintain the validation of the information, you must try to make all corresponding data changed the same time in a designate way. This is usually impossible.

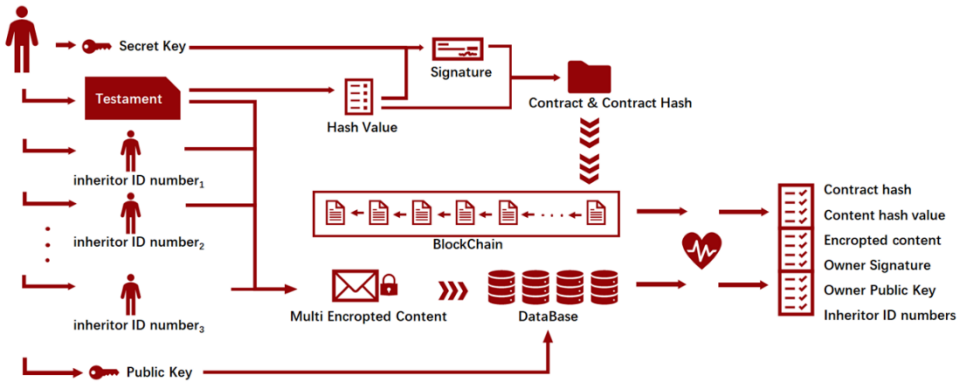
3.2 Pain point 2: Time-consuming versus Highly Efficient Online System

Block chain is deployed on information systems, you can finish all things with a terminal. The development of information technology and user interface design makes it easier and easier for customers to operate machine and manage personal data. Blockchain as a new technology is also an helpful interface to greatly improve the efficiency of information process.

3.3 Pain point 3: Complex Rules in Testaments versus Smart Contract

Blockchain technologies, especially ethereum and the technologies later, can deal with complex logical processes of transactions, which make the applications on blockchain much “smarter”. Developers use a piece of code called smart contract to implement commercial contract, business logic and service in various fields like copyright protection, reserved evidence, and so on. Smart contract is a piece of logical code, which is more abstract and concise than tangible materials and also it works with high reusability, simplifying the process to set a legal testament.

4 System structure and mechanism



4.1 User lifetime

User registration, and begins to fill in the will.Fill in the will including content and the relatives public key, the system automatically uses the user's private key to sign. The sig and the hash of the will content is will be stored in block chain. Obtain the smart contract address, and save it to our database with the encrypted content using the relatives’ public key.

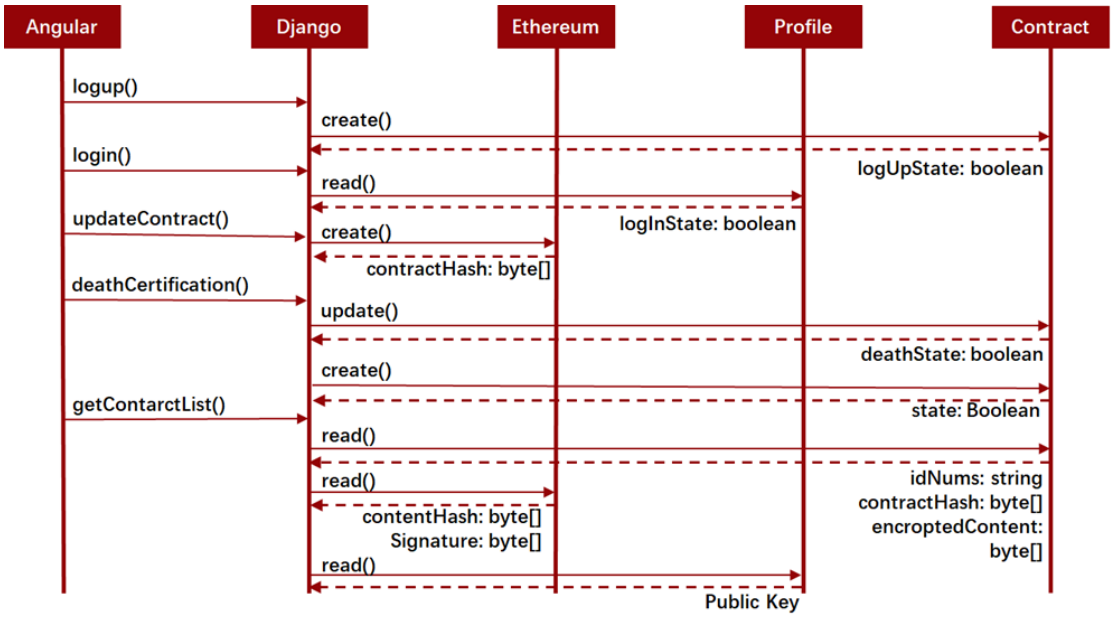
4.2 After the user is dead

Inheritor can access contract address, content hash, encrypted file, signature, public key, relatives' ID

Confirmation content: Decrypt the content with the successor private key one by one, and finally compare it with the content hash

Confirm signature: use user signature, user public key, user's testament content hash for confirmation

5 System sequence diagram



6 Front-end structure

We use Angular 8.3.15 as the development framework for the user interaction part of the website.

6.1 Angular Component

In the Angular Project, we maintained 7 components, which are responsible for the layout on the webpages, the data transactions with back end coded by Django by RESTful APIs. Besides, the project build an Angular module called app-routing.module, which make the routing mechanism and the data in which working.

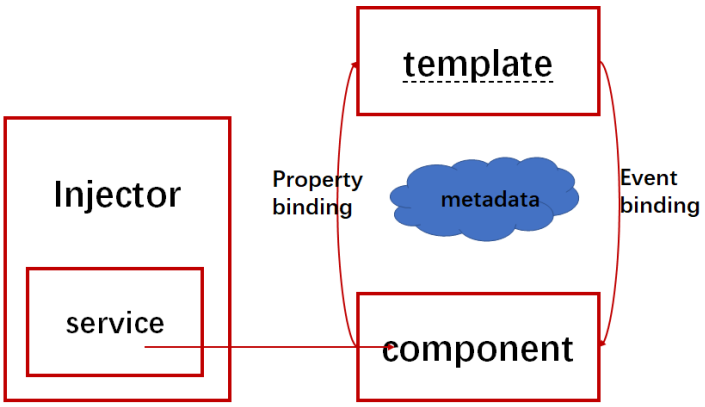
6.2 Angular Service

In the front-end part, there are two Angular services working in the whole process, which are HTTP.service and share.service. The first one, HTTP.service package the service by which front end communicates with back end in HTTP method. The service also make it easier to test the front end project by using mock address pointing to a local folder containing mock data. The second one, Share.service is a service supported by a kind mechanism of Angular called subscribe. The service is built for make data generated in a child component flow in its parent component. Angular

does not have this kind of built-in function in its template, so we build it ourselves in the service.

6.3 Angular Templates

Angular templates generate the webpages, which are coded in specified format in component.html files. With Zorro UI toolkits, the development of webpages layout, forms and so on are much more interactive for users.



Source from: https://angular.io/guide/architecture

7 Front-end requests

The requests in front end project are maintained by Angular ts files, actually, ts files handle complex logic responsible for each webpage, including Angular injection mechanism, component initial process, web page model construction and so on.

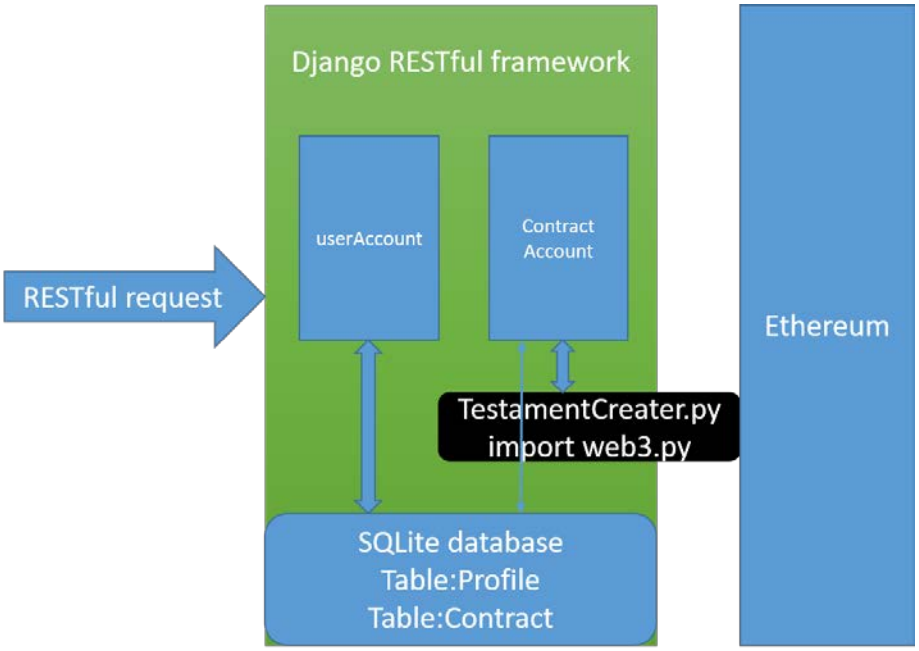
7.1 Log-in and log-up and death-certification

Front end project generate the form to collect information from users to check identity of users and serve for users in designate method and webpages. Log-in, log-up and death-certification requests will provide the web serve with information about user identity and handle the response.

7.2 update-contract and get-contract-list

Front end project will collect the content of testament and the relative information of testament then generate specified formatted file, which will be sent to web serve through encryption protocol. When relative contracts return from serve, Angular will simple put the designate data list into a Zorro graphic user interface container to show the list of relative contracts in a friendly way for users.

8 Back-end structure



8.1 Brief

We use Django restful-framework as our back end framework, including two apps, userAccount and countractAccount. All information from these two apps is recorded in the default SQLite database which has two tables: Profile and Contract. Profile table has all the information of our user, and the Contract table has all the information of created smart contracts, using a user's IdNum as foreign key and the address of a contract as primary key. After determine the smart contract on remix, we just simply interact with it using web3py package. To put our contract on the ether, we just call constructor and the functions of the solidity contract.

8.2 userAccount

Create user using the ProfileCreate class, which handles POST and GET request. After checking the validation, create a new data into the Profile table.

Another function is to check log in, if the user is in the database than return True.

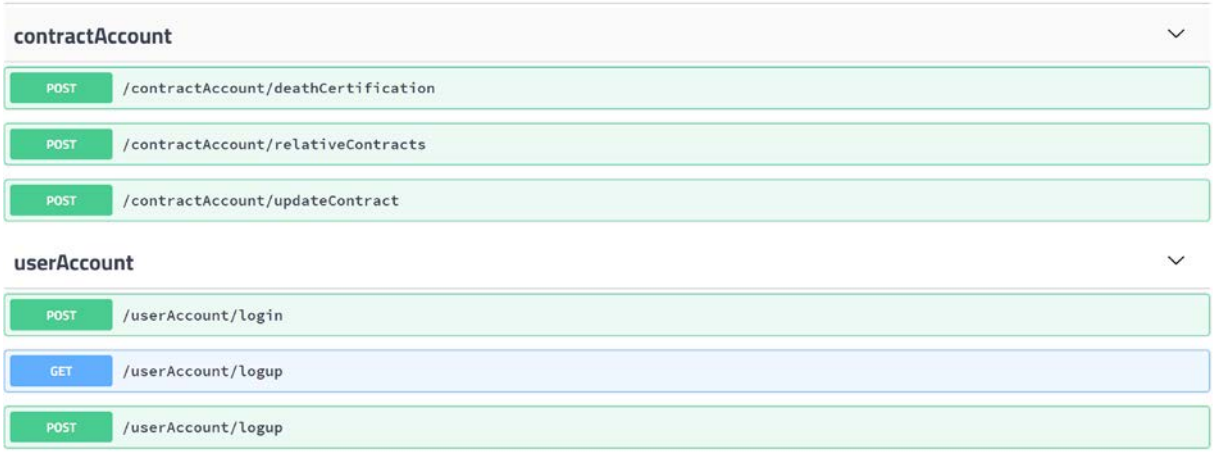
8.3 contractAccount

Create contract data using the UpdateContract class, which handles POST and GET request. After checking the validation, create a new smart contract with TestamentCreater class, encrypt the data and store everything into Contract table.

After the owner of a testament has passed away, all the information will be accessible by the assigned relatives. The relatives can see all the testaments that is related to them in a page, which will call our

RelativeContracts class. The RelativeContracts will query all the information from database and interact with the smart contract to get data.

9 Api



9.1 logup/

{
"idNum": id number of the creator,
"psw": password of the account,
"publicKey": publicKey of the creator,
"email": email of the creator
}

returns:

{
"code": "1" //1 success 0 fail
}

9.2 login/

{
"idNum":id number of the user,
"psw":pass word of the user
}

returns

{
"code": "1" //1 success 0 fail
}

9.3 updateContract/

{
"content": the testament content ,
"idNum": id number of the creator,

"relativeIdNums":[
id number of the relatives

],
"private_key": private key of the creator
}
returns
{
"code":"1", // or 0
"contractHash": address of the smart contract
}
9.4 deathCertification/
{
"idNum":id number of the dead,
}
returns
{
"code":"1" // 0, if no relative contract
}
9.5 relativeContracts/
{
"idNum": id number of the relative
}
returns
{
"contracts":[
{
"contractHash":"",
"encroptedContent":"",

"contentHash":"",
"ownerSig":"",

"ownerPublicKey":"",

"idNums":[
{
"idNum":""

```
                    },
                    {
                        "idNum":""
                    }
                ]
            },
            {
                "contractHash":"",
                "contentHash":"",
                "encroptedContent":"",
                "ownerSig":"",
                "ownerPublicKey":"",
                "idNums":[
                    {
                        "idNum":""
                    },
                    {
                        "idNum":""
                    }
                ]
            }
        ]
}
```