House Rental Decentralized Application

EECE 571G: Blockchain Software Engineering

Instructor: Dr. Zehua Wang

Final Report

Kwan, Hiu Hong (Terry) 57663866

Hongrong Zhang 49990088

Danni Zhao 63903827

Jiayu Huang 48389043

Guowei Guan 45929304

Department of Electrical and Computer Engineering

The University of British Columbia

April 10, 2022

Table of Contents

1. Background

1.1 Blockchain

Blockchain technology is a decentralized technique of accounting records that enables the storage and tracking of transaction data without the need for central authority oversight or intervention. The blockchain idea was first presented by Nakamoto[1] in 2008 as the foundation for a Bitcoin cryptocurrency financial system that keeps its value and transaction history essentially in perpetuity. After that, the use of blockchain technology has moved beyond cryptocurrencies to include various types of data.

1.2 Ethereum and Smart Contract

Ethereum is a blockchain-based platform that aims to enable developers to quickly build consensus-based apps. It comprises a blockchain with an integrated turing-complete programming language, Solidity, that enables anybody to create smart contracts and decentralized apps with their own ownership rules, transaction formats, and state transition methods.[2]

A smart contract is a computer program that automatically executes or enforces predefined contract conditions on the blockchain. The contract's terms and conditions are instantly inserted into the program line.

1.3 Decentralized Application

A decentralized application (DApp) is a decentralized network-based application that integrates back-end smart contracts with a front-end user interface. It may be deployed on peer-to-peer (P2P) or blockchain networks. For example, PayPal, Google Pay, and Cash App are softwares that operate on computers that are connected to a P2P network in which numerous individuals consume, supply, or seed content, or run these functions concurrently[3].

[1] S. Squarepants, "Bitcoin: A peer-to-peer electronic cash system," SSRN Electron. J., 2008.
[2] V. Buterin, "A next generation smart contract & decentralized application platform," 2015.
[3] J. Frankenfield, "Decentralized Applications (dApps)," Investopedia, 08-Feb-2022. [Online]. Available: https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp. [Accessed: 19-Mar-2022].

In the context of cryptocurrencies, DApps operate on a blockchain network in a public, open-source, decentralized environment that is free of any one institution's control or intervention. For instance, a developer may design an Instagram-like DApp and deploy it on the blockchain, allowing any user to publish a message. Once published, these statements are irreversible, even by the app's author.[4]

2. Introduction

A traditional web-based business application generally consists of a database, a back-end module, and a front-end module. If the server fails, the whole program becomes inaccessible. Additionally, the only component of the system that is partially visible to the end-user is the front end. However, the database is hidden from the user. The whole system is controlled and owned by a single individual or business; the data may be tampered with on purpose by the owner; the data can be buried or provided in part, or hackers can attack the system and manipulate the data.[5]

On the other hand, DApps enable developers to build fully accessible applications that operate independently of any organization. A decentralized blockchain protects and controls the program's data and code, preventing the application from having a single point of failure. Additionally, the data is safe and secure since it is saved on the blockchain. A user must contact a blockchain node in order to interact with a smart contract. If a node fails, the other nodes compensate.

[4] J. Frankenfield, "Decentralized Applications (dApps)," Investopedia, 08-Feb-2022. [Online]. Available: https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp. [Accessed: 19-Mar-2022].
[5] M. Farnaghi and A. Mansourian, "Blockchain, an enabling technology for transparent and accountable decentralized public participatory GIS," Cities, vol. 105, no. 102850, p. 102850, 2020.

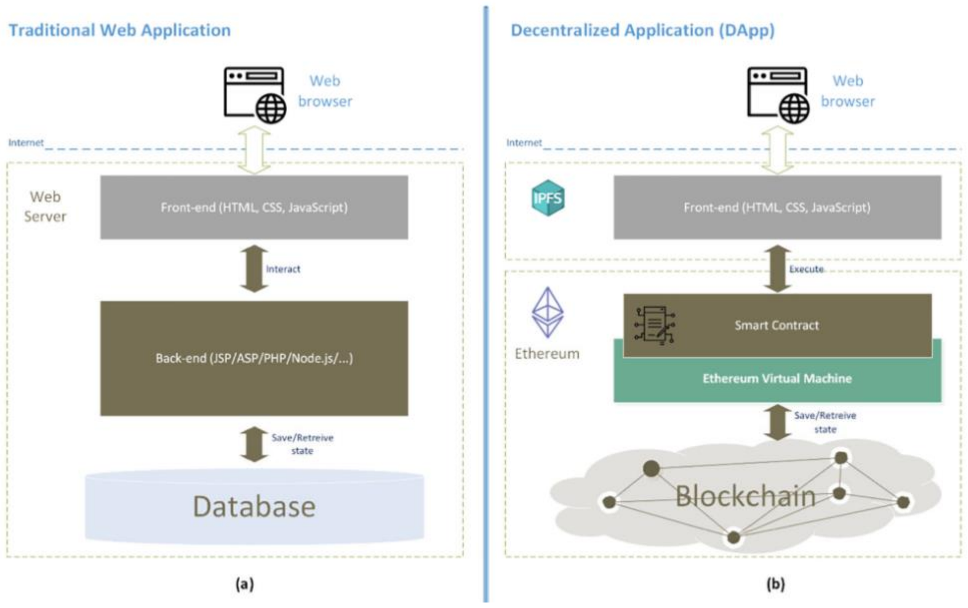


Figure . Traditional Web Application vs. DApp

A variety of house rental apps are accessible in the market. To join in the majority of them, contact information, addresses, and other personally identifiable information must be provided and public, which may result in data breaches and spam. The issue stated above may be mitigated or resolved by using decentralizing technologies to house rentals. The potential tenant's uploaded background information can only be viewed by the target landlords.

Without the assistance of a realtor or a real estate firm, people may create a contract between individuals in the house rental DApp[6]. A landlord uses the DApp to advertise his or her house on the Internet and obtains the rental agreement via communication with a renter who supplies background information. After that, the landlord creates a Smart Contract. After the landlord and tenant sign the Smart Contract, it is deemed completed when the tenant puts bitcoin into the blockchain system.

In addition, the house rental DApp is always available to serve clients wishing to engage with it once the smart contract is placed on the blockchain. Therefore, hostile actors are unable to conduct a distributed denial-of-service attack on it. Secondly, the house rental DApp is censorship-resistant such that no one organization on the network has the

[6] M. Farnaghi and A. Mansourian, “Blockchain, an enabling technology for transparent and accountable decentralized public participatory GIS,” Cities, vol. 105, no. 102850, p. 102850, 2020.

ability to restrict users from running the DApp, or accessing data from the blockchain. Lastly, the DApp has a verifiable behavior that rental smart contracts may be examined and guaranteed to execute in a predictable way without relying on a central authority.[7]

3.1 Function Design

When a user initially accesses the House Rental DApp, he or she can register as a landlord or a potential tenant. Users can only register an account when they are connected to MetaMask. The abilities of both roles are described below.

3.1.1 Landlord:

- Upload house information
- Change the house information
- Get the tenant-candidate list
- Check the candidates' background
- Send the contract to the ideal candidate

3.1.2 Potential Tenant:

- Upload personal background
- Change the personal background
- View list of house information
- Check the availability of the house
- Show interest to landlord
- Sign the contract with the landlord
- Cancel the match with the landlord

[7] "Introduction to dapps," ethereum.org. [Online]. Available: https://ethereum.org/en/developers/docs/dapps/. [Accessed: 19-Mar-2022].

3.2 Use Cases

The figure below shows the flowchart of using the House Rental DApp.

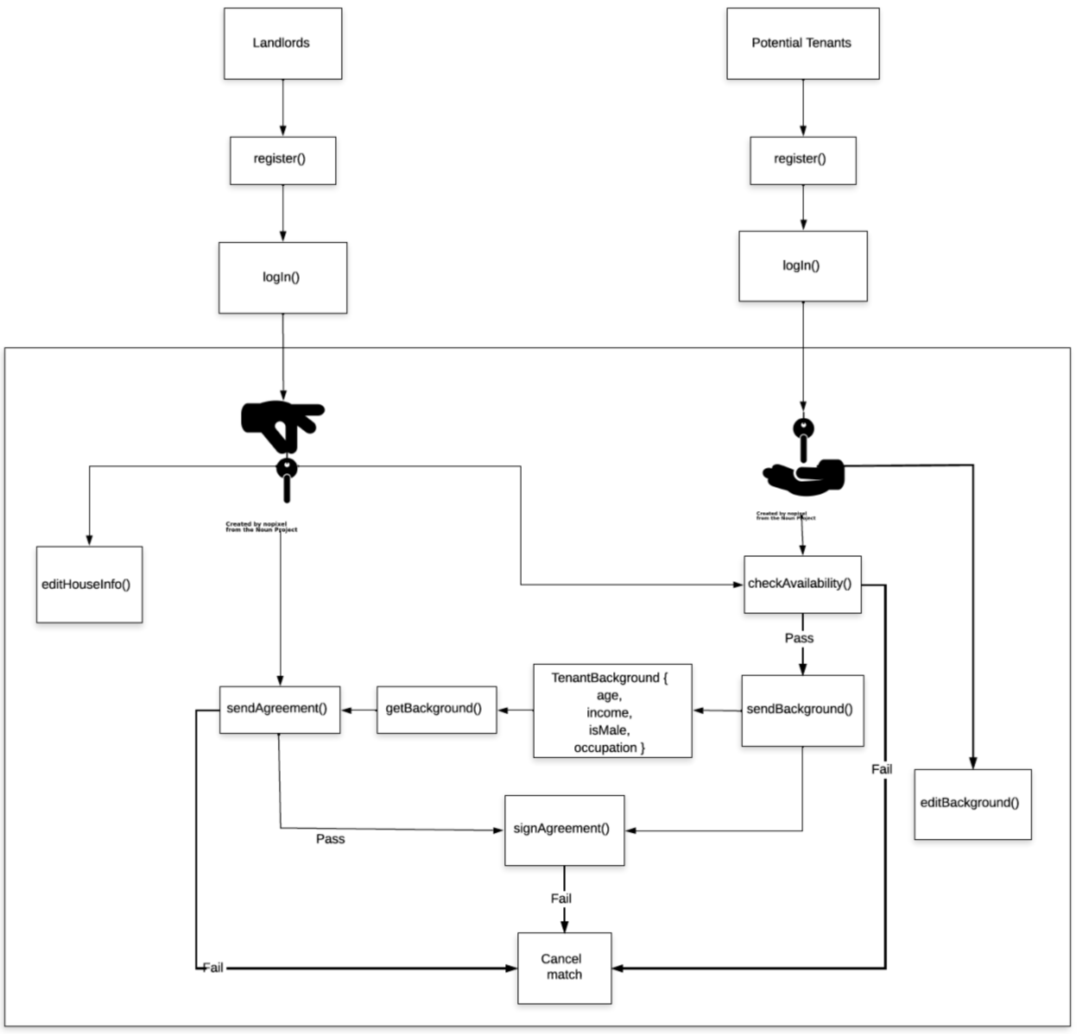


Figure 1. House Rental DApp Use Cases

4. Smart Contract

One can find our smart contract code in /contracts folder on our project github

https://github.com/EECE571G21W2ProjectGroup3/TermProject

4.1 Data Structure

Several user-defined mappings and user-defined structs are constructed in the smart contract to store user information.

4.1.1 User-defined mappings

```
1.mapping(address=>User) public users;
2.mapping(uint256 => address) public idUsersMap;
3.mapping(address => TenantBackground) public backgrounds;
4.mapping(address => HouseInfo) public houses;
5.mapping(address => address[]) public landlordTenantsMap;
6.mapping(address => address[]) public tenantLandlordsMap;
7.mapping(address => address) public matchedPairs;
```

1. User mapping, given the MetaMask account, retrieves the given user's account information stored in the struct user (explained below).
2. Given the user's unique ID, retrieve the user's MetaMask account address associated with this ID.
3. Given the MetaMask account address of a registered tenant, retrieve the tenant background associated with this account.
4. Given the MetaMask account address of a registered landlord, retrieve the house information associated with this account.
5. Given the MetaMask account address of a registered landlord, retrieve a list of MetaMask account addresses of tenants who are in the waitlist of the house.
6. Given the MetaMask account address of a registered tenant, retrieve a list of MetaMask account addresses of landlords who have sent an agreement to this tenant.
7. Given either the tenant or landlord address of a signed agreement, return the signed landlord/tenant associated with this agreement.

4.1.2 User-defined structs

```
1.struct User {
        uint256 userID;
        string name;
        string password;
        string userType; //either "tenant" or "landlord"
        string phoneNumber;
        string email;
}

2.struct HouseInfo {
        string houseAddress;
        string rental;
        string description;
        string period; //Available from DD/MM/YYYY to DD/MM/YYYY
        bool isHouseAvailable;
}

3.struct TenantBackground {
        uint age;
        uint income;
        bool isMale;        // true->male, false->female
        string description;
}
```

1. User struct: upon finishing registration, the required registration information of the user will be sent to the smart contract and stored in the User struct; an unique ID will also be assigned to the registered user by the system.
2. HouseInfo struct: landlord-side struct used for storing house information after the landlord has updated the house info.
3. TenantBackground struct: tenant-side struct used for storing tenant background information after the tenant has updated the background info.

4.2 Functions

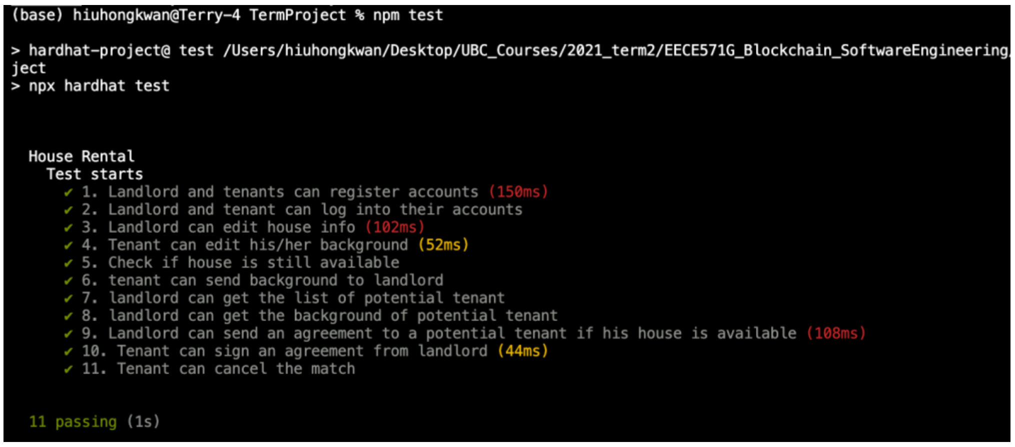
Here we listed all functions we implemented for our smart contract and their effects.

- **register()**: A potential landlord or tenant can register an account on the platform using his/her meta mask account
- **logIn()**: A registered user can log into the platform with his/her username and password
- **editHouseInfo()**: A landlord can add or change the information of his/her house on the platform
- **editBackground()**: Tenant side function, used for tenants to modify their background information to store in the TenantBackground struct.
- **checkAvailability()**: Tenant side function, used for tenants to check the availability of an existing house rental.

- **sendBackground()**: Tenant side function, used for tenants to send their background to a landlord when the rental availability is true.
- **getPotentialTenants()**: This function enables only landlords to get the list of potential tenants.
- **getPotentialLandlord()**: This function enables only tenants to get the list of potential landlords after receiving agreements.
- **getBackground()**: The landlord can get the tenants' background from the potential tenant list.
- **sendAgreement()**: The landlord can send an agreement to the ideal matched tenant.
- **signAgreement()**: The matched landlord and tenant can sign the agreement together.
- **cancelMatch()**: This function is triggered automatically when the "checkAvailability" or "sendAgreement" is false. Also, when the tenants click the "cancel match button" at the web page, the match will be canceled.
- **resetMatch()**: This function will reset the matchedPair mapping to null once tenant decides to start a new search

4.3 Test Result

Below is a screenshot of the test cases that were passed.



```
(base) hiuhongkwan@Terry-4 TermProject % npm test

> hardhat-project@ test /Users/hiuhongkwan/Desktop/UBC_Courses/2021_term2/EECE571G_Blockchain_SoftwareEngineering
ject
> npx hardhat test



  House Rental
    Test starts
      ✓ 1. Landlord and tenants can register accounts (150ms)
      ✓ 2. Landlord and tenant can log into their accounts
      ✓ 3. Landlord can edit house info (102ms)
      ✓ 4. Tenant can edit his/her background (52ms)
      ✓ 5. Check if house is still available
      ✓ 6. tenant can send background to landlord
      ✓ 7. landlord can get the list of potential tenant
      ✓ 8. landlord can get the background of potential tenant
      ✓ 9. Landlord can send an agreement to a potential tenant if his house is available (108ms)
      ✓ 10. Tenant can sign an agreement from landlord (44ms)
      ✓ 11. Tenant can cancel the match


  11 passing (1s)
```

Figure 2. Test Result ScreenShot

4.4 Hardhat Framework

Hardhat is a development environment to compile, deploy, test, and debug your Ethereum software. It helps developers manage and automate the recurring tasks that are inherent to the process of building smart contracts and DApps, as well as easily introducing more functionality around this workflow. This means compiling, running and testing smart contracts at the very core.

In our project, we constructed our smart contract based on an initialized framework that Hardhat provided to us to test the basic functionalities of our smart contract using built in commands such as `hardhat compile` and `hardhat test`.

4.5 Web3.js API

Web3.js ethereum javascript API is the most popular API that is used for connecting React.js frontend to the backend smart contract. Web3.js is a JavaScript library that allows us to communicate with the Ethereum blockchain. It turns our React application into a blockchain enabled application.

We created our connection from our web server to the blockchain by constructing an Alchemy web3 instance into our frontend code using HTTP.

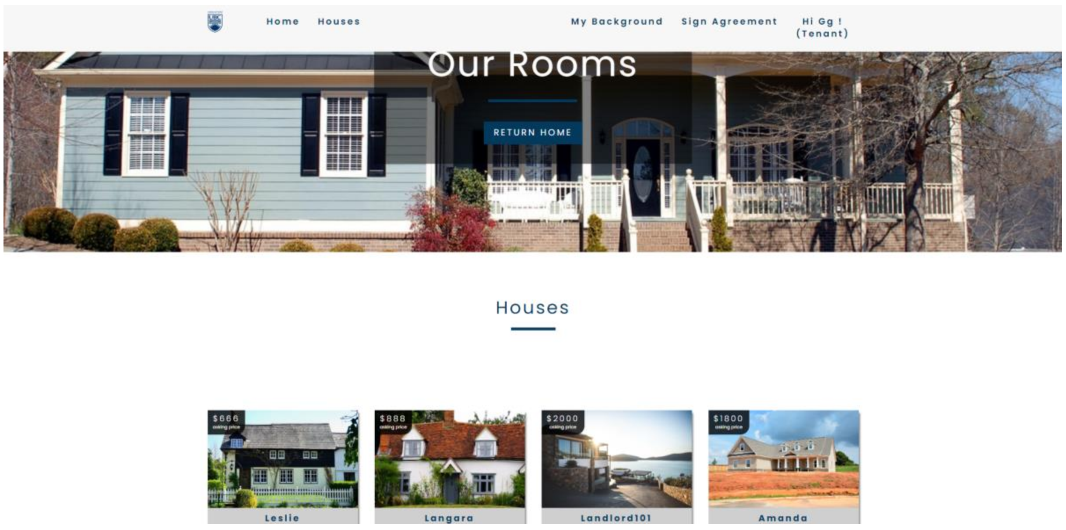
```
const web3 = createAlchemyWeb3(`https://eth-
ropsten.alchemyapi.io/v2/${contractDetails.ALCHEMY_API_KEY}`);
```

Now we could deploy our smart contract using web3.js provided function to create a smart contract instance by retrieving the ABI and address of the contract from the blockchain.

```
const deployedContract = new web3.eth.Contract(
    contractDetails.abi,
    contractDetails.contractAddress
);
```

With the instances in hand, now we could utilize all the functions that are residing in the smart contract and interact with the frontend. An example is shown below.

```
try {
      result.result = await deployedContract.methods
        .register(name, password, userType, phone, email)
        .send({ from: sessionStorage.getItem("walletAddress") });
} catch (err) {
      alert(err.message);
      result.error = err.message;
}
```



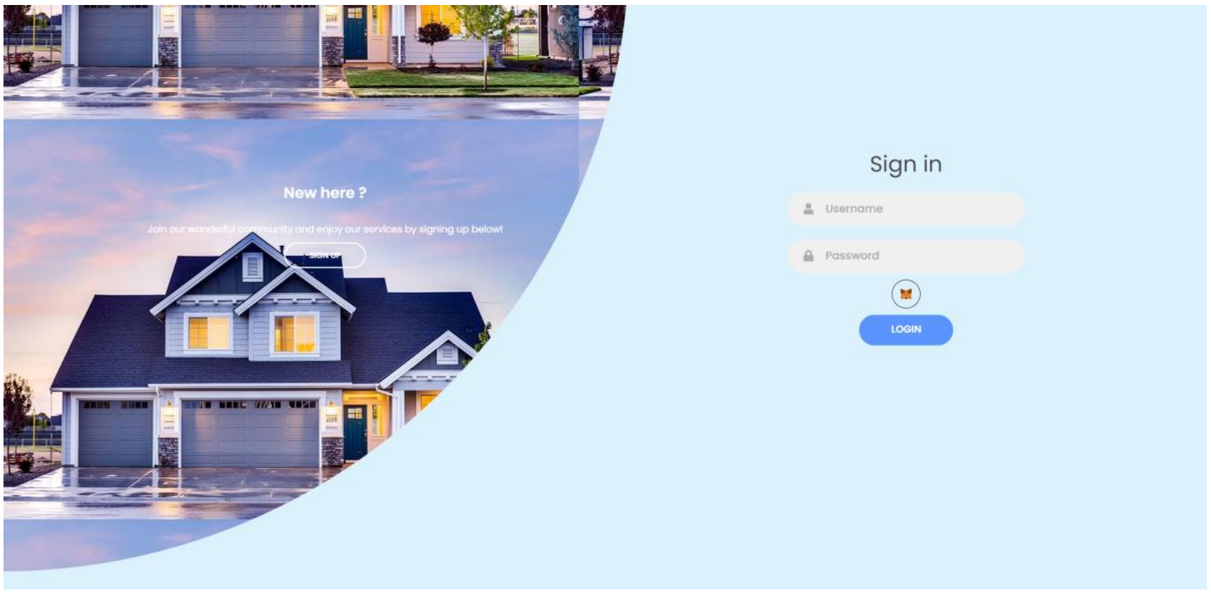
5. User Interface Design

We used React as our frontend framework and Metamask with the Web3 library to manipulate the blockchain. For the design of the webpage, we mainly follow the template on GitHub provided by ShiroWorks.[8]

5.1 Sign in/up form

Different from our original design, we managed to put our sign in and sign up form on the same page, forms can switch from one to another by double sliding.
Figure 3 gives a brief demonstration of our login page for both the landlords and residents, where they can either log in or be slidered to the sign up page if they don't have accounts already. Users can login with their registered username, password and link to their metamask wallet with the button below the check box.

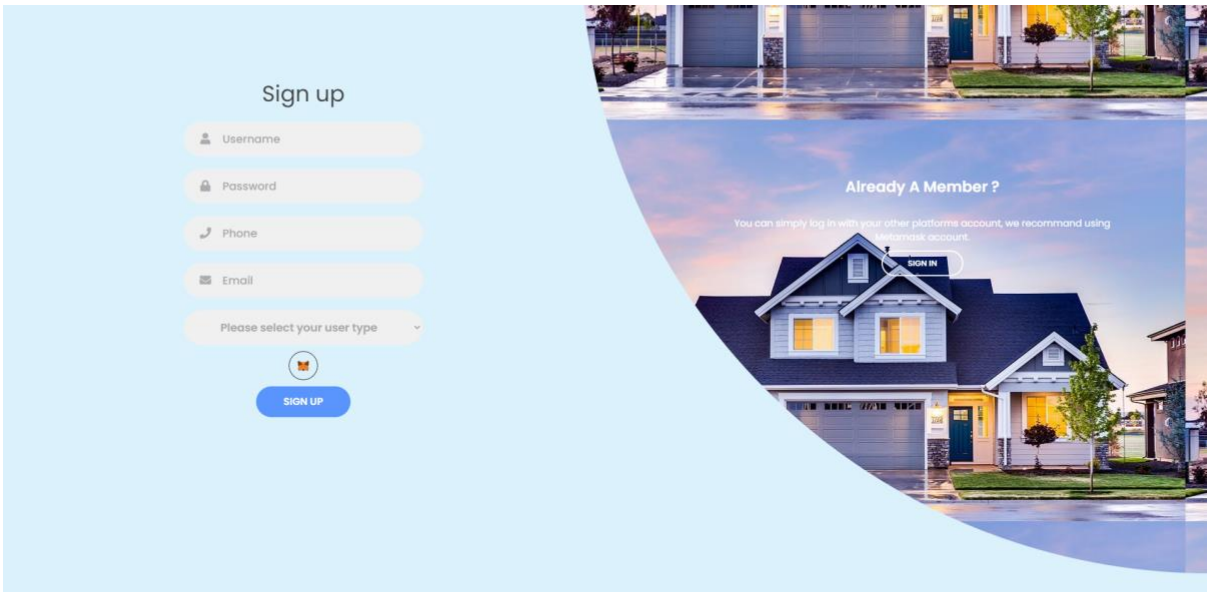
[8] ShiroWorks. (n.d.). *ShiroWorks/react-real-estate-website*. GitHub. Retrieved April 11, 2022, from https://github.com/ShiroWorks/React-Real-Estate-Website



Join our wonderful community and enjoy our services by signing up below!

Figure 3. Sign In Page

Figure 4 illustrates our setup page for our users; it can also be slided to the login page. Users are required to register with their username, password, phone number, email address and their metamask wallet. Different user types are required when the user finishes their setup, as the landlord and tenant shall have different content in the home page. They will be redirected to a different home page when they login to their landlord/tenant account.



You can simply log in with your other platforms account, we recommand using Metamask account.

Figure 4. Sign Up Page

5.2 The overlay animation

We have 4 smaller screens/boxes inside the main component: the sign in form, the sign up form, the sign in overlay, and the sign up overlay. Also, at one moment in time users can see either:

1. The sign in form alongside the sign up overlay
2. The sign up form alongside the sign in overlay

In the overlay panels we have some text and a button, by clicking it will bring up the other combination of screens and vice-versa.

We have the following layers for the overlay component:

1. The overlay-container: this container will show our visible area (more on this below) at a certain moment in time. It is set in a CSS file with a width of 50% of the total container's width.
2. The overlay: this container has a double width size so it's taking the full width of the main container.
3. The overlay-panels: they are the divs which hold the actual content (the text and the button) on the screen. They both have a position of absolute, meaning that one can position them wherever we want in the overlay component. One of the panels is positioned to the left and the other one is positioned to the right, both having a width of 50% of the overlay component.

Basically we have two containers, the form-containers where each having a width of 50% and a position: absolute. We move both of them at one time from the left to the right, and when they get behind the overlay-container from above (when these are moving along) we immediately set the z-index value so the sign up form will move on top of the sign in form, it also works vice-versa.

5.3 Metamask wallet connection

MetaMask provides a global API embedded with websites visited by its users at window.ethereum. This API allows our websites to request users' Ethereum accounts, read data from blockchains the user is linked to, and suggest that the user sign messages and transactions. In our sign in/up pages we created a button that will get access to a unique Ethtereum public address, with which users can start sending and receiving ether or tokens. It allows our websites to request user login, load data from blockchains the user has a connection to, and suggest the user sign messages and transactions. We have an asynchronous function which will make a call to MetaMask and get the information we requested. We want to acquire information about our account number, Chain ID, Account and ETH balance. We create a function which takes the parameters and stores the data to the localstorage so that we don't lose if the user refreshes the browser. We connect to Metamask and retrieve wallet information but if we refresh we could lose information in the UI. In order to tackle this information we will use builtin react hook useEffect to check whether we have this object in the window global object or not.

5.4 Tenant account pages

Considering the content of a tenant account has different functionality from the landlord's one, we made two types of pages for each type of users. Figure. 5 illustrates the home page for tenants. Residents can enter their requirements to match with the house information in our system, or they can simply choose the one that is available on this rental page.

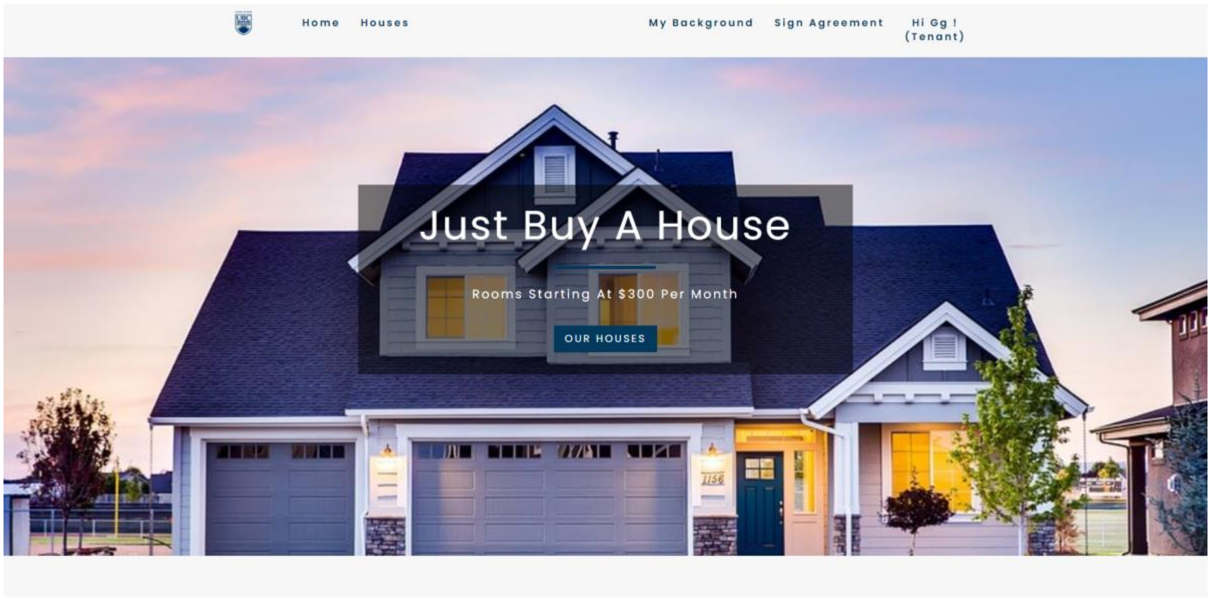
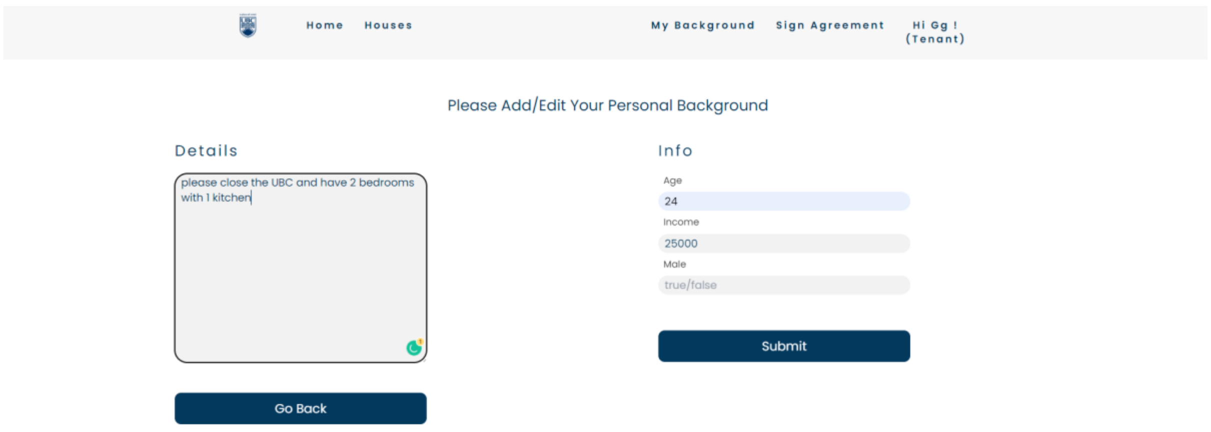


Fig. 5 Home page for tenant account

Tenant type users can submit their basic information and house requirements to our website, such information will be used for matching up available houses on our website and viewable to the landlord when the tenant expresses an interest in a specific house. This page is shown in figure .6.



Please Add/Edit Your Personal Background

Details

please close the UBC and have 2 bedrooms with 1 kitchen

Info

Age: 24

Income: 25000

Male: true/false

Figure.6 Tenant background

In the tenant account, users can view the available houses listed on our website, which is submitted by the landlord from their account. Such content is demonstrated in figure.7.

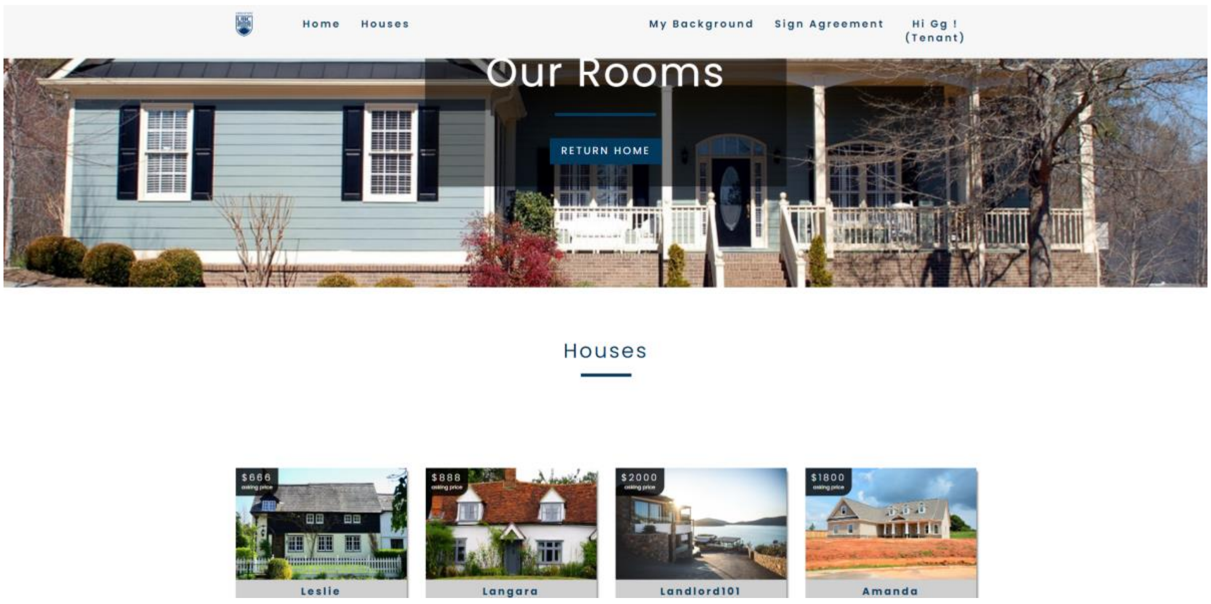
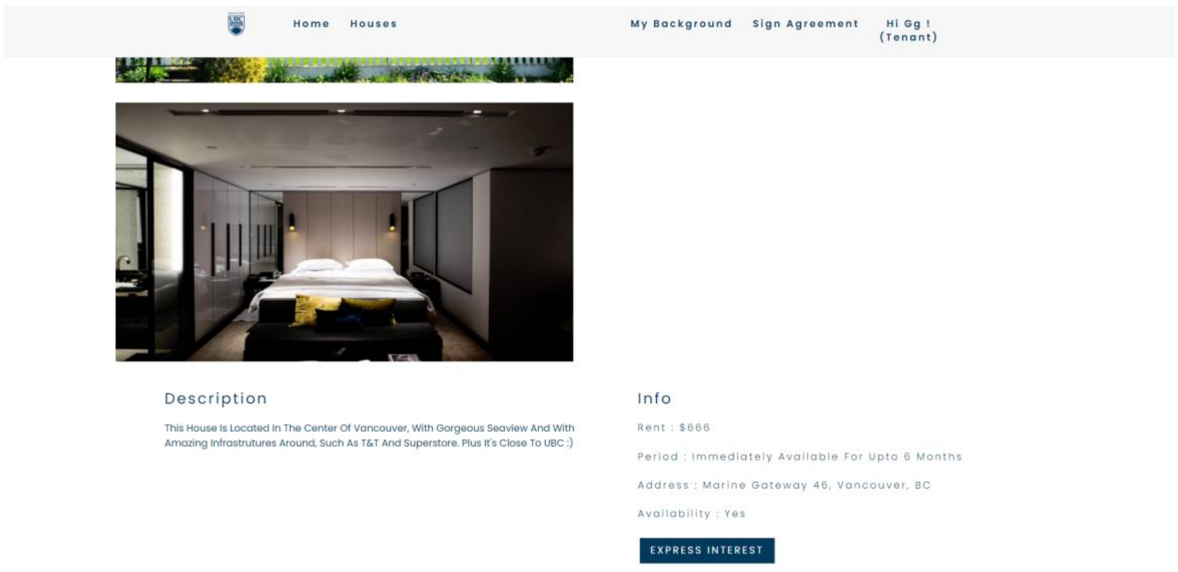


Figure. 7 Available houses list

If the user shall find any one of the listed houses meets his/her requirement, he/she can send an interest request to our website(fig. 8), which shall be redirected to the landlord of this specific house, if the landlord agrees with this request from his/her lists of interest request, he/she can agree with the request and send an agreement to the tenant on our website.



Description

This House Is Located In The Center Of Vancouver, With Gorgeous Seaview And With Amazing Infrastrutures Around, Such As T&T And Superstore. Plus It's Close To UBC :)

Info

Rent : $666

Period : Immediately Available For Upto 6 Months

Address : Marine Gateway 46, Vancouver, BC

Availability : Yes

Figure. 8 House information page on houses list

5.5 Landlord account pages

On the landlord home page, users can upload house information, receive interest requests from the tenant, and send the agreement back to the interested tenant. The landlord homepage is illustrated in the figure. 9.

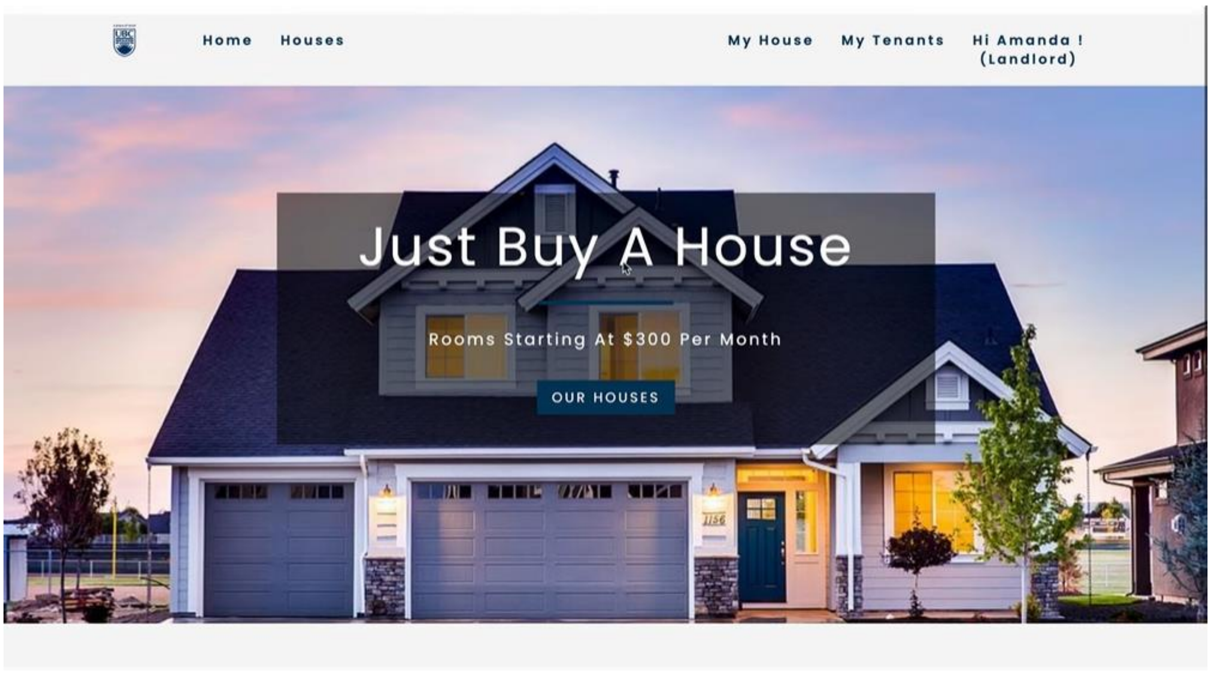
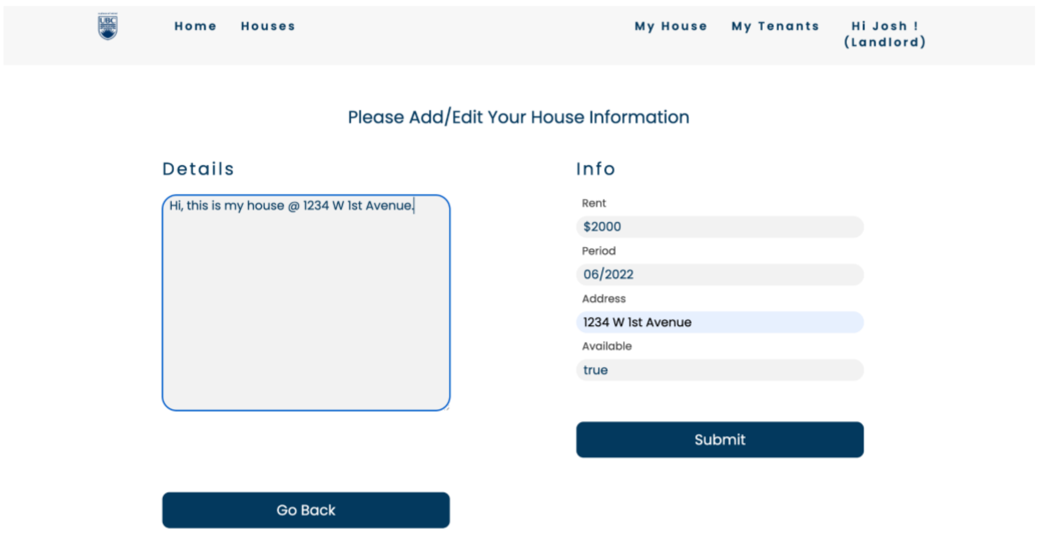


Fig. 9 Landlord account home page

Landlords are able to submit their houses information through "My House" page(fig.10) to our houses pool which will be displayed on the "Houses" page on the tenant terminal.

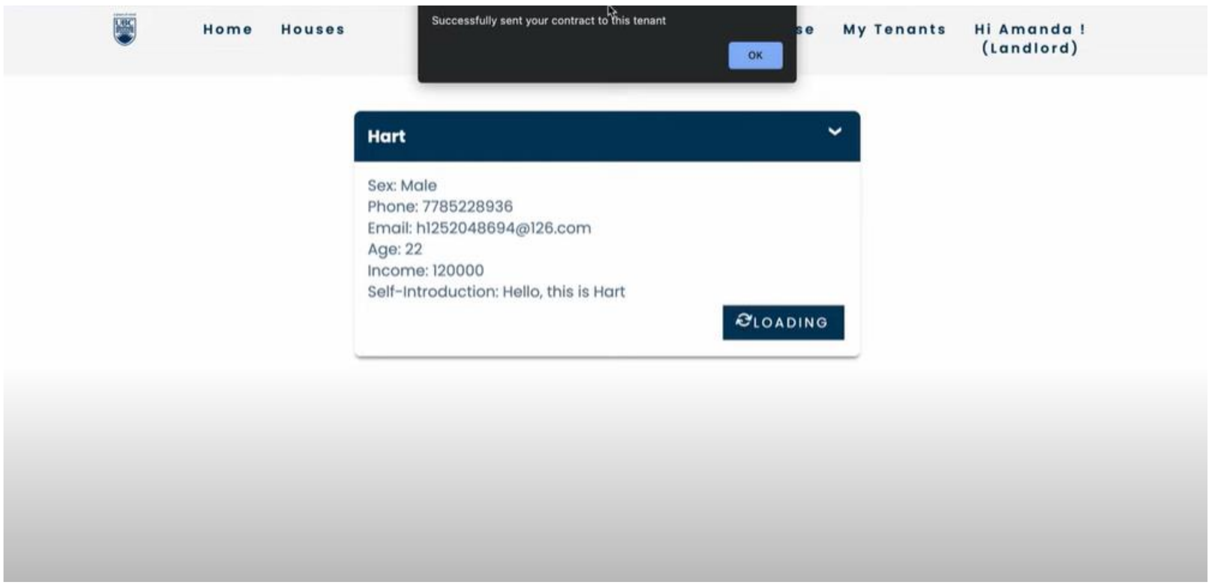


Hi, this is my house @ 1234 W 1st Avenue.

Rent: $2000
Period: 06/2022
Address: 1234 W 1st Avenue
Available: true

Fig. 10 House information submission page

In the “My Tenant” page, landlord-type users can view the interest request for their houses with the basic information of the tenant, and the landlords can send an agreement to one of the tenants(fig. 11).



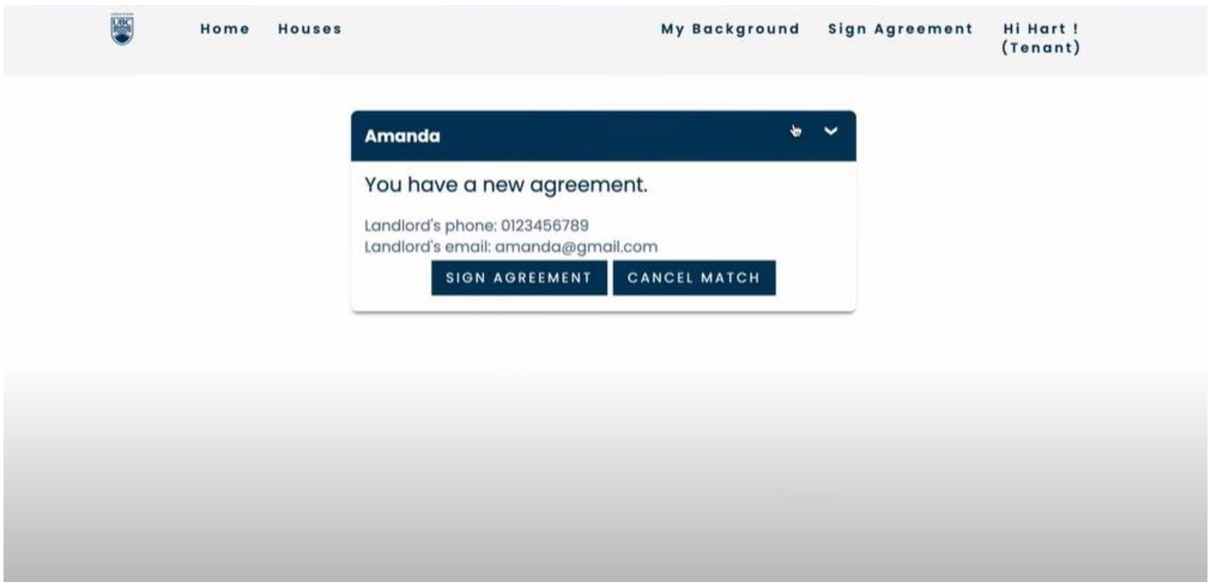
Successfully sent your contract to this tenant

Hart

Sex: Male
Phone: 7785228936
Email: h1252048694@126.com
Age: 22
Income: 120000
Self-Introduction: Hello, this is Hart

Fig . 11 Interested request from tenant in “My Tenant”.

On the tenant side, the agreement will pop up in the “Sign Agreement” page(Fig. 12).



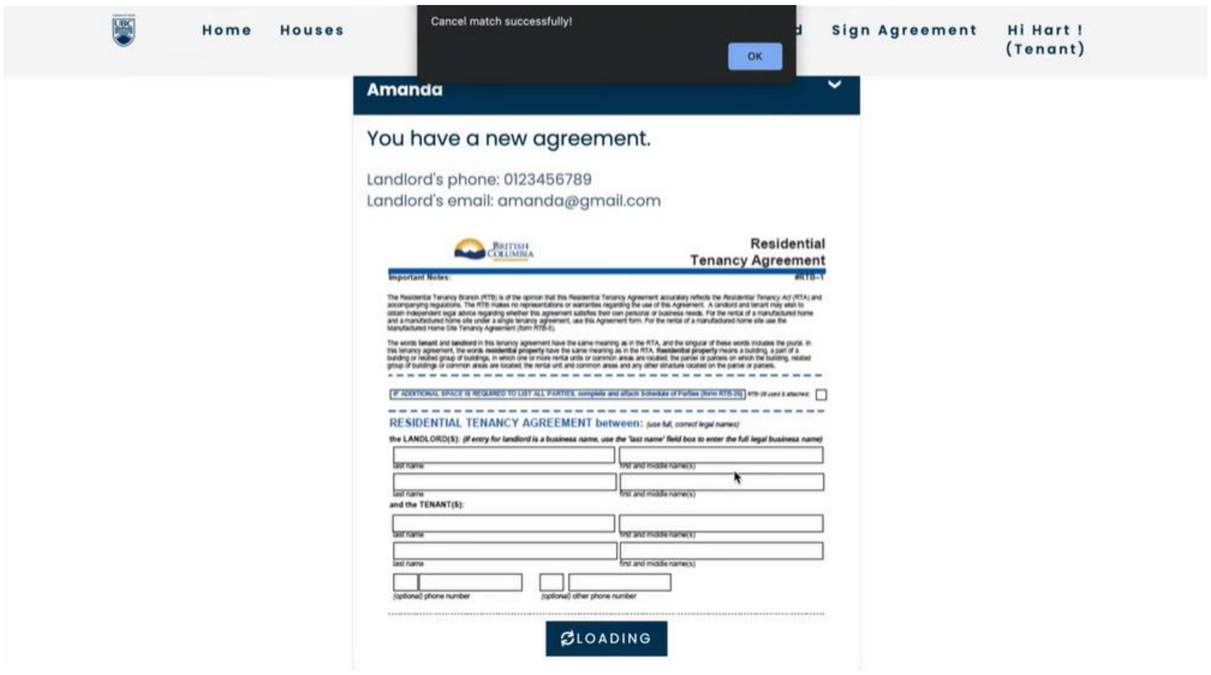
Amanda

You have a new agreement.

Landlord's phone: 0123456789
Landlord's email: amanda@gmail.com

Fig. 12 new agreement notification in tenant's "Sign Agreement" page.

After the landlord and tenant has made an agreement, they will be redirected to the contract sign up page, where they can fill in their information and sign the contract to terminate this process(fig. 13).



Cancel match successfully!

Amanda

You have a new agreement.

Landlord's phone: 0123456789
Landlord's email: amanda@gmail.com

Residential Tenancy Agreement

Important Notes: #RTB-1

[illegible]

RESIDENTIAL TENANCY AGREEMENT between: (use full, correct legal names)

the LANDLORD(S): [illegible]

last name | first and middle name(s)

last name | first and middle name(s)

and the TENANT(S):

last name | first and middle name(s)

last name | first and middle name(s)

(optional) phone number | (optional) other phone number

Fig. 13 Contract sign up page for tenant and landlord

6. Conclusions and Future Work

We designed a DApp for house rentals in this project. Users can join the platform as either a landlord or a tenant and browse the platform's housing resources. A landlord-type user can upload his or her own house to the site, view a list of tenants who have expressed interest in it, and send contracts to the target tenants. The target tenants are assigned on a first-come, first-served basis. Once the contract agreement is co-signed, the house will be removed from the platform's available inventory. A tenant-type user can upload his or her background to the platform, and the landlord will review the eligibility if the tenant expresses interest in the landlord's house.

This project's objective is to investigate the capabilities of blockchains and decentralized applications. At the moment, the application is a tokenless decentralized application, which may make participation difficult due to a lack of incentives. In the future, we may create and publish our own token to enable users to participate in the rental platform's development.

Another critical feature that was overlooked during the creation of this programme is the security of the data (particularly user passwords) stored on the blockchain; the application currently lacks hashing techniques to prevent data leakage. To protect our users' data, hash algorithms such as MD5 and SHA-256 should be incorporated into our algorithm.

7. References

[1] "Introduction to DApps," *ethereum.org*. [Online]. Available: https://ethereum.org/en/developers/docs/DApps/. [Accessed: 19-Mar-2022].

[2] J. Frankenfield, "Decentralized Applications (DApps)," *Investopedia*, 08-Feb-2022. [Online]. Available: https://www.investopedia.com/terms/d/decentralized-applications-DApps.asp. [Accessed: 19-Mar-2022].