Animadopt: An Short-Distance Animal Adoption Dapp

Li Ju, Runze Wang, Shanny Lu, Shijun Shen

### Abstract

Stray animal mobile applications are popular all over the world to resolve the stray animal issues. However, according to the current situation, those issues haven't been effectively resolved in most cities. So, there is still a long way to go in resolving stray animals’ problems. To help solve these issues, in our project report, an short-distance animal adoption decentralized application, Animadopt, was designed and programmed using React.js with Javascript as the main front-end language and Solidity as the back-end programming language. The application enables users to post, share and adopt the stray animals on the Google Maps APIs in the form of markers and secures the transaction as users in Dapps can transact directly with each other using cryptocurrencies without third-party disturbing. This work provides a relatively convenient tool for pet-lovers to put their joint efforts to help stray animals find their new home.

**Keywords**: Animal Adoption, Blockchain, Decentralized application

### 

### 

[**Abstract**](#_uecgpsoe6z9z) **1**

[**Introduction**](#_koaftm7y997t) **4**

[**Background**](#_6uip59xusds9) **4**

[Previous Work in Animal Adoption](#_mwgru1dho9x0) 4

[Animal Adoption Application Using Blockchain](#_l60tupv347j) 5

[Introduction to blockchain technology](#_i1jwp88aa9w4) 5

[**Design**](#_xh1jg4v40np9) **6**

[3.1 Front-end Design](#_tk09yhjy70o) 7

[3.1.1 Stray Animal Map](#_67iri0j8h0bg) 7

[3.1.2 Transaction](#_mxdomvil41ev) 8

[3.1.3 User](#_dqdzqocudtem) 9

[3.1.4 Prototypes](#_frjcqk8bl8wl) 12

[3.2 Back-end Design](#_9zahj9e55a9f) 13

[3.2.1 Design of storing data](#_lwxesrbw6spu) 13

[3.2.2 Design of events](#_1b3gtrplhppy) 15

[3.3 Agent Design](#_7m4orufuw4ra) 16

[3.4 Overall Design](#_9zahj9e55a9f) 16

[3.5 Use Cases Design](#_9zahj9e55a9f) 16

[3.5.1 User Use case:](#_9w8e6rlg4db1) 17

[3.5.2 Post Animal Information Use case:](#_dvahardmuqi7) 17

[3.5.3 Transaction Use Case:](#_mxaqcdkhfcfy) 17

[**Implementation**](#_ci7u8b58u23p) **18**

[4.1 Design Implementation](#_148cev78wni0) 18

[4.1.1 Front-End](#_uecuvawrlkpi) 18

[<App/>](#_yv4gwyctg1o) 18

[<Nav/>](#_286yj6knn599) 18

[<CurrentLocation/>](#_c3ijyitxnjvw) 18

[<MapContainer/>](#_84u68cjuze6m) 19

[<SignInPage/>](#_x0ymw8win8r9) 20

[<SignUpPage/>](#_15qv55d1g2qu) 21

[<TabMenu/>](#_g21o74fipqa0) 22

[<UserInfoPage/>](#_8ipuz5255got) 22

[<UserProfilePage/>](#_5vb16h60qsnv) 23

[<UserTransPage/>](#_v5vy7j48m7kd) 24

[<PostInfoPage/>](#_c6412dvitld2) 25

[<AnimalInfoPage/>](#_lud1y8ei4wax) 26

[<OrderConfirmPage/>](#_56aut2fftx51) 26

[4.1.2 BackEnd](#_4d1bp32j33x0) 26

[4.1.3 Agent](#_9bidgtokzamz) 28

[4.2 Problems Encountered and Relative Solution](#_148cev78wni0) 29

[**Testing**](#_92e08uxzovd5) **31**

[5.1 Front-end Testing](#_9jaid1c3qtgt) 31

[5.2 Back-end Testing](#_9jaid1c3qtgt) 32

[**Discussion & Future Improvement**](#_vuavo124yxnh) **33**

[**Conclusion**](#_opw6snwbpuok) **34**

[**Acknowledgment**](#_ii8nmkgt40z7) **35**

[**References**](#_f0o975ww92uy) **36**

### 

### Introduction

Stray animals have many negative impacts on the city environment and human health. For example, they can cause noise pollution, feces garbage and traffic accidents. Also, stray animals have become a bigger threat to human health and lives, because stray animals can transfer and spread Rabies, which is a deadly disease. According to the World Health Organization (WHO), there are more than 200 million stray animals worldwide, and 55,000 people die from rabies every year.

According to Animal Shelter Statistics, more than 78,000 cats, 28,000 dogs, and 11,000 animals of other species were received by Canadian shelters in 2019. While the majority of these intake animals were categorized as stray animals (40% of dog intake and 50% of cat intake). And the second largest intake group are animals abandoned by their owners (35% of dog intake and 31% of cat intake). Regarding the outcomes of these shelter dogs and cats, adoption is regarded as the largest outcome category, with 45% of dogs and 60% of cats adopted in a new home in 2019 [1]. All data above illustrates that there exists a huge potential and great demand of building an animal adoption decentralized application, which can help stray animals to find their new home.

By leveraging Dapp’s inherent security property, as users in Dapps can transact directly with each other using cryptocurrencies without third-party disturbing, the transaction security can be greatly maximized. We proposed a stray animal adoption Dapp, aiming to provide a more convenient and secure approach for warm-hearted pet-lovers to take action in helping stray animals find their new home, or even missing pets to find their original hosts.

### Background

#### Previous Work in Animal Adoption

Many works have been conducted to assist animal adoption, especially the traditional mobile app. Edoption, is one of those current animal adoption applications. It allows users to browse through available stray animals nearby. Users can swipe right to like a pet or adopt it [2]. Raenu (2016) also develops a prototype of stray animal app for Malaysians, aiming to reduce the number of stray animals in Malaysia by introducing an app that is convenient for the public to use [3]. Examples of existing applications are Animal-Free, which is an application that allows users to scan the product barcode or daily usage product such as soap to make sure the ingredients are not animal-derived [4]. This is a way to support saving animals by saying no to animal-derived products. Another application called Reswalk is a charity that raises funds for animals based on a unique idea of raising funds while walking. Other applications such as AADR allow users to report animal abuses as long as sufficient evidence such as photos are provided.

#### Animal Adoption Application Using Blockchain

However, there are still some limitations in these traditional database animal adoption applications. First, mutability. Databases for centralized applications take great risks of data tampering. In this situation, the security of user accounts is threatened. Second, transactions could not be transparently viewed by all users. For animal reporters, they could not easily tell whether the adopters are reliable or not. Also, if there is something wrong, it is hard to track the money. Blockchain can solve the pain points previously described.

#### Introduction to blockchain technology

Nowadays, blockchain technology attracts massive attention from different industries including finance, healthcare, supply chains etc [5]. Compared with traditional databases, blockchain saves committed transactions together in a chain of blocks [6]. Decentralization applications, also could be called Dapps, execute based on a blockchain or P2P network instead of a single computer. Thus, Dapps have key characteristics of blockchain such as anonymity, persistency and decentralisation [6].

Besides Bitcoin, there are various adoption of the technology due to Ethereum. This is because Ethereum Virtual Machine, smart contracts and Solidity language provide convenience for cryptocurrency project developments.

Blockchain becomes a popular topic in the development field because of special characteristics pointed out above. Firstly, decentralized blockchains are immutable so that the security of transaction accounts will be improved. Each node which also refers to each computer holding a blockchain owns complete data records. If one of the transaction records is tampered, all other nodes will simply find the node with incorrect information by cross-reference each other. Secondly, all transactions could be transparently viewed by anyone. Therefore, it is easy to trace the stolen cryptocurrency. Thirdly, blockchain technology reduces cost. For example, as a trade based on currency exchange, banks typically are required to verify a transaction. However, with the help of blockchain, the fees people pay for third-party verification could be eliminated. Finally, blockchain supports an efficient transaction environment. To settle transactions through central authorities might take a couple of days, nevertheless, applying blockchain could reduce the transaction time to a few minutes.

### Design

To help pet-lovers post stray animals information that they found, our proposed stray animal adoption Dapp can provide them a convenient approach to post the stray animal information and adopt nearby appropriate animals .

The target users of our designed Dapp would be generally animal lovers. More specifically, we can divide them into three groups, based on their positions in the animal adoption transaction process:

1. Stray animal reporters - People who can post stray animal information;
2. Information viewers, including adopters - People who can adopt the stray animal;
3. (Future work) Animal adoption organizations.

It is worth mentioning that the short-distance transaction process is our main focus, as we invoke Google Maps APIs as a main application interface.

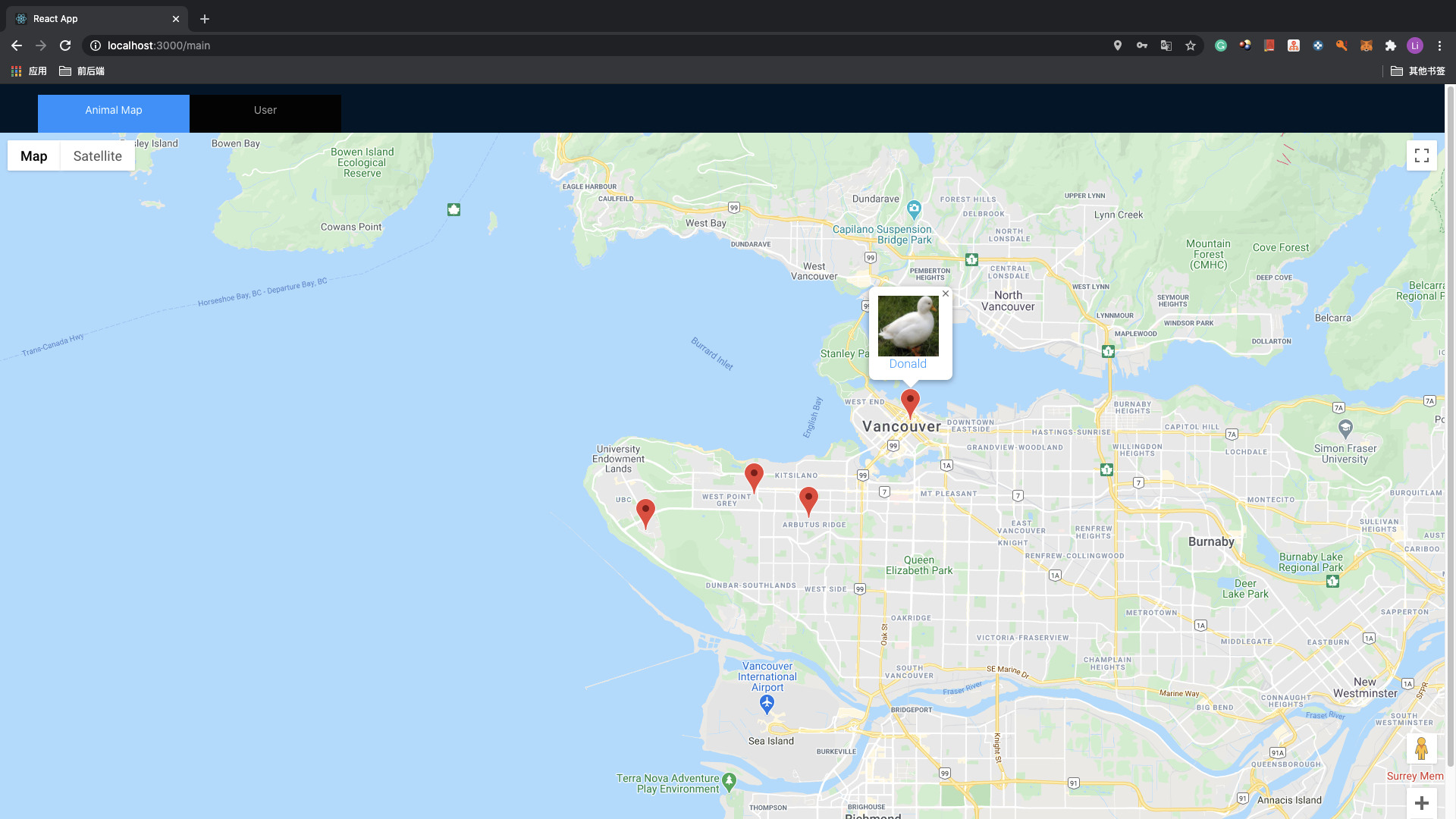
Our proposed short-distance animal adoption Dapp can mainly support the following functionalities:

1. Google Maps APIs are invoked to visualize where stray animals have been found and posted. Thus, users can check the nearby available post stray animal information;
2. Users can post animal information, including position, pictures, description etc., on the map;
3. Users can check the map for the stray animal nearby to adopt it;
4. Users can click the pop-up points on the map to check brief descriptions (photos), and further order confirmation is required;

#### 3.1 Front-end Design

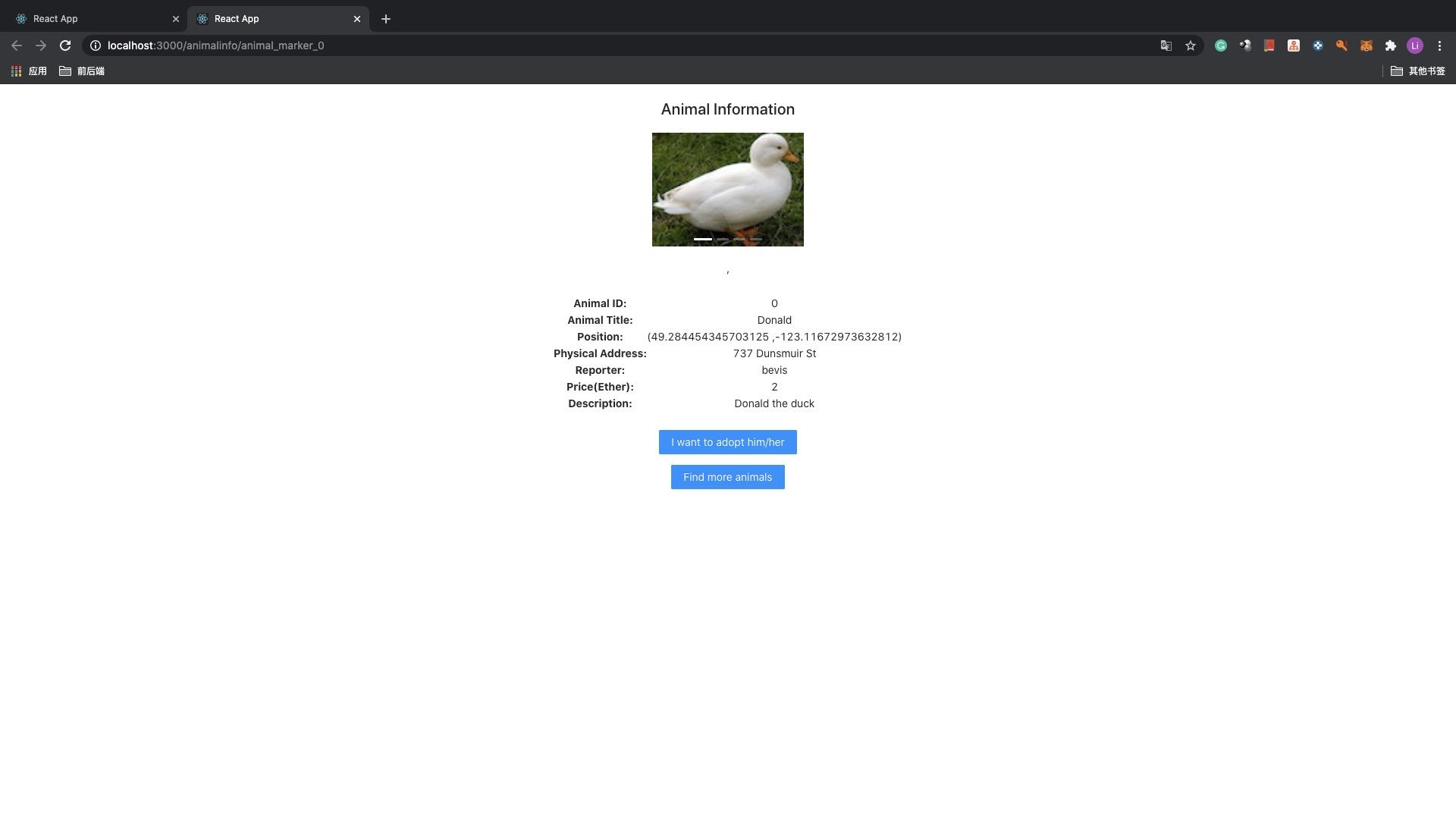
By analyzing the supported functionalities, we divided them into three parts: map showing the locations of stray animals, animal-related transactions, and user activities. The overall structure of this front-end design is based on these three branches.

##### 3.1.1 Stray Animal Map



To present users with a map including all the information of stray animals, we invoke Google Maps APIs for animal position visualization. Every animal posted on our site can be represented by a mark in the real-time map, with a preview image popping up for users to browse.

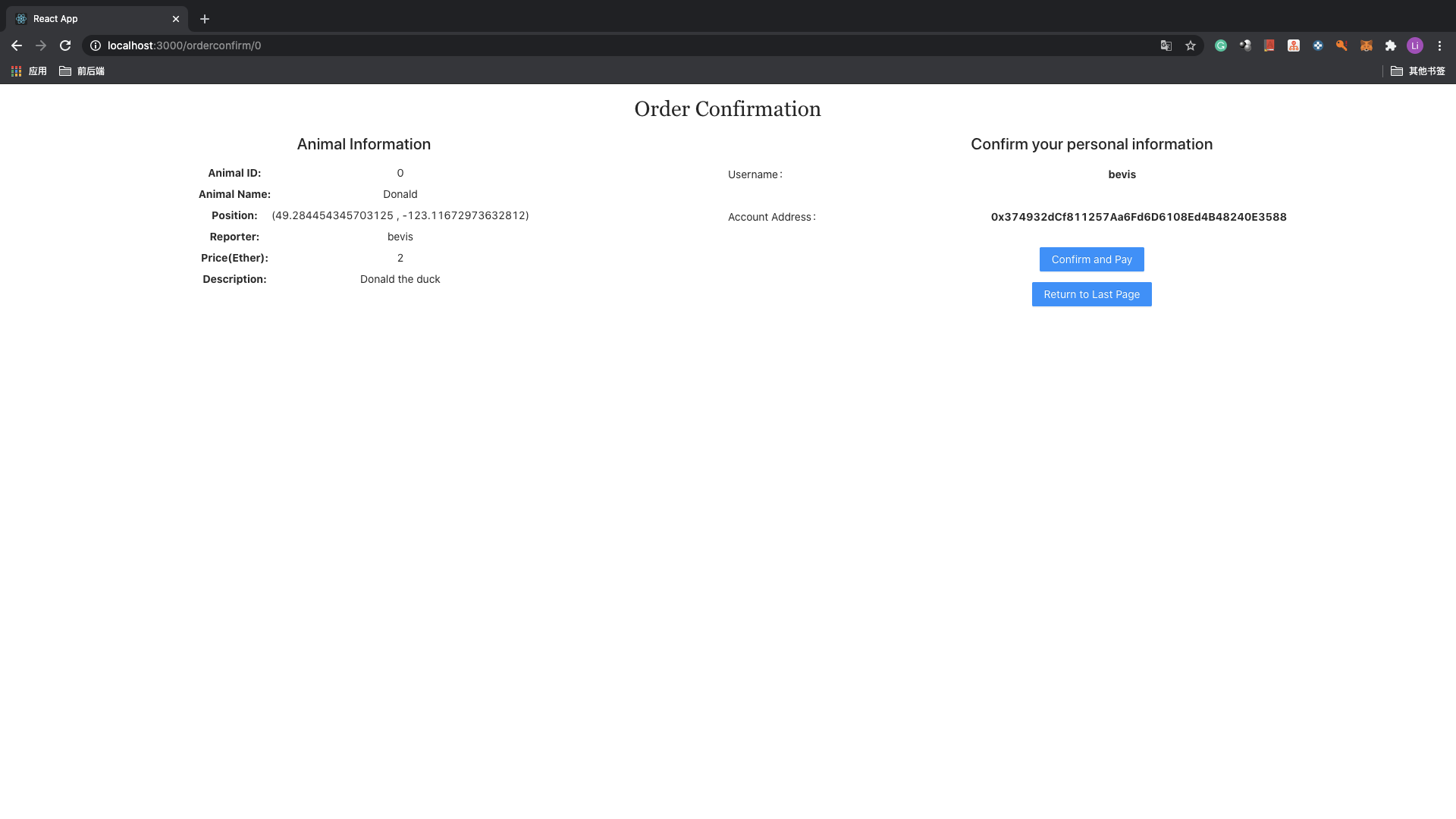
There are two tabs on the top of the map, users can click one of them and the corresponding web page will be displayed.



Once a user clicks on a marker, the user will be redirected to a page containing specified animal information related to the marker, with its photos displayed in carousel mode, location, price in ether, and people who post this information.

If a user is interested in adopting this animal, he/she can choose to adopt this animal by clicking the “I want to adopt it” button. Otherwise, they can also return to the main page to browse more options.

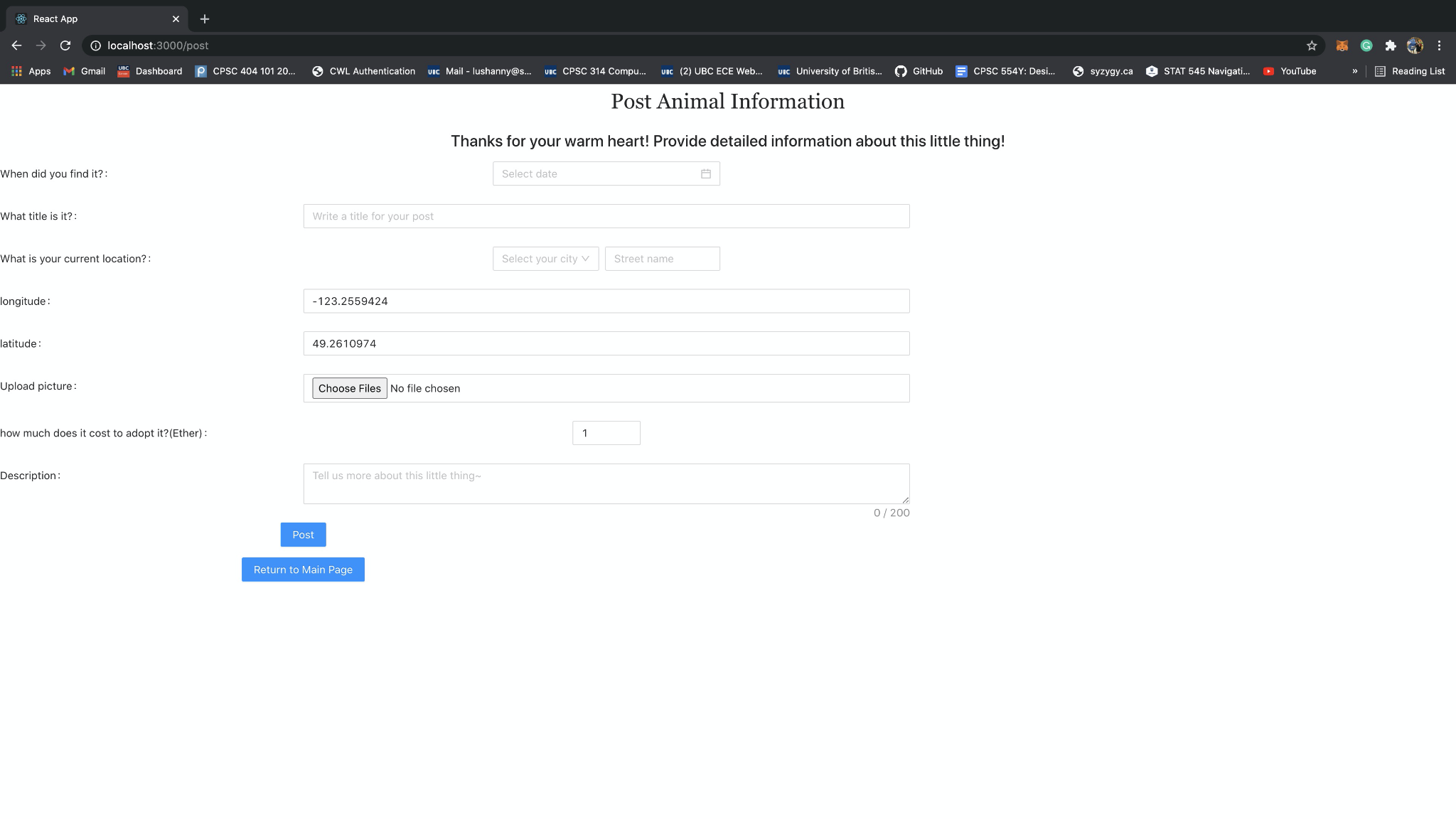
##### 3.1.2 Transaction



If a user decides to adopt an animal, he/she will be linked to the order confirmation page. In this page, After reviewing everything he/she needs to know about this transaction, the user can choose to confirm the payment, which will awake metamask to deal with the ether transfer part, or to change his/her mind and go back to the animal information page.

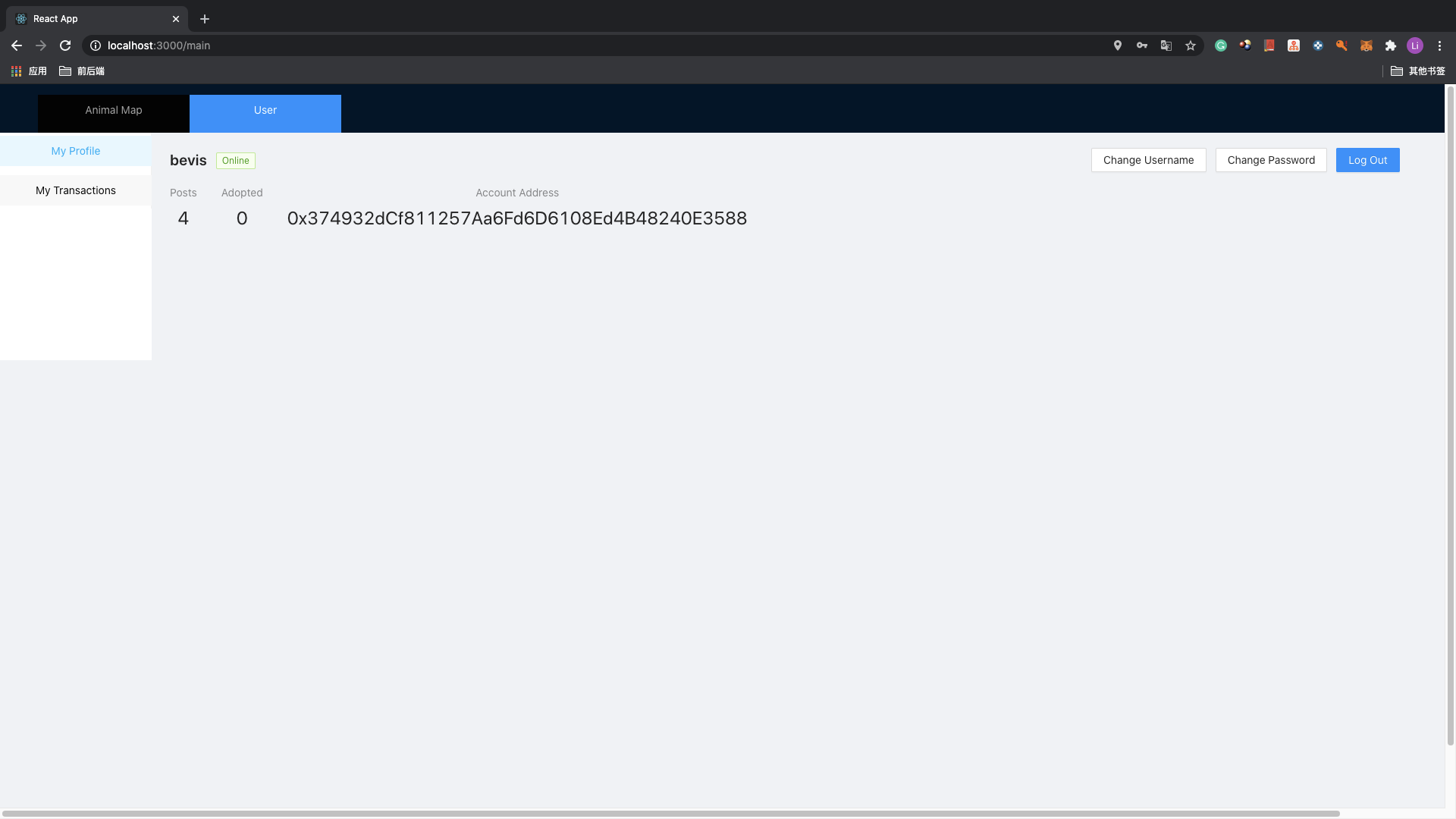
##### 3.1.3 User

In our design, users activities consist of two parts: reviewing user profiles and posting stray animals.

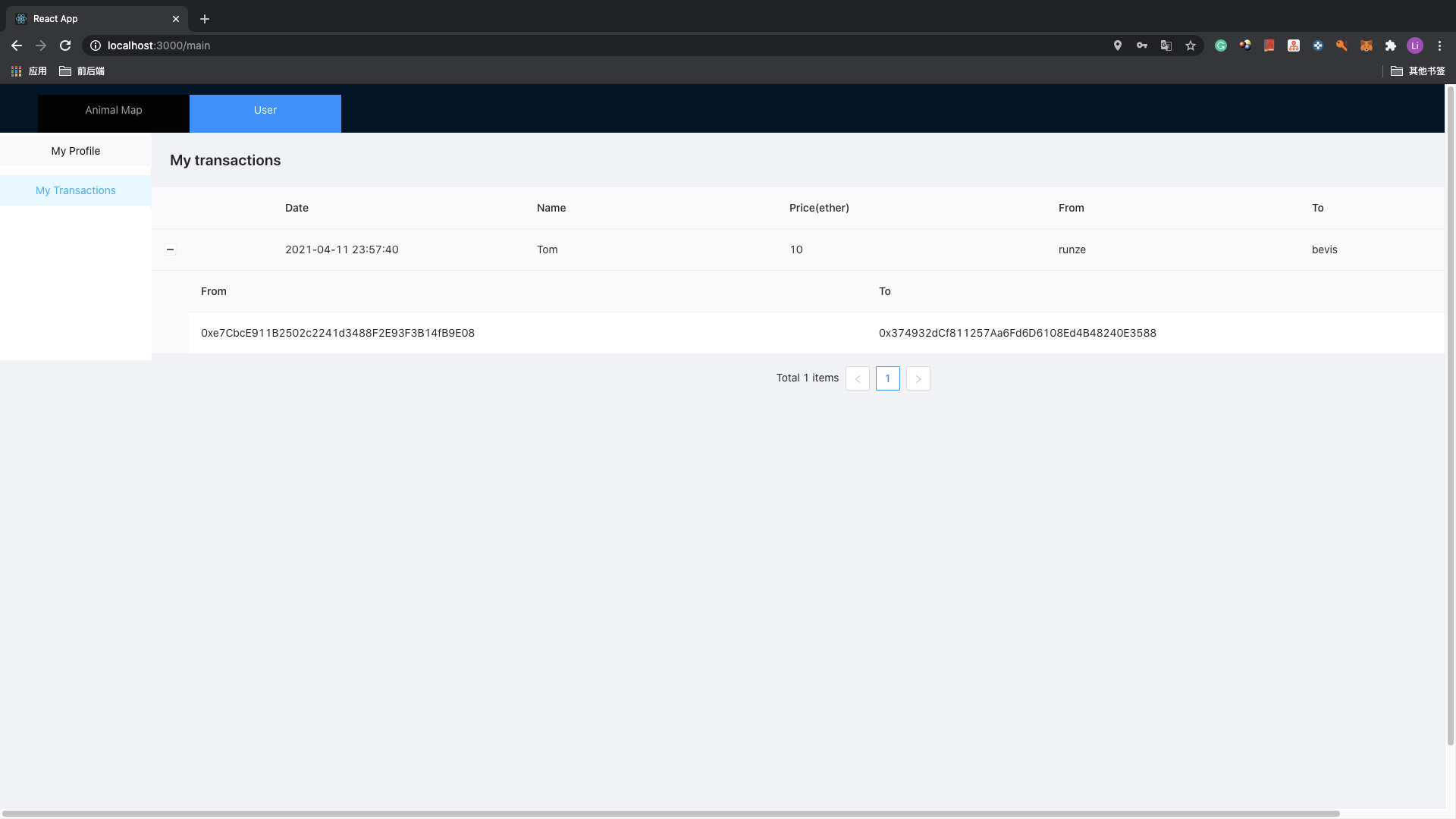


This page is designed for users to post any animal he/she has found and look for potential adopters. In this page, user should enter as many details as they can about the stray animal, including but not limited to: the time when they find the animal, their current location(displayed in latitude and longitude), the price they want to put on the animal(unit: ether), photos the user takes about the animal, and all other necessary information for other animal lovers to know more about this animal.

As for reviewing user information, we introduce a user profile page and a user transaction history page. To provide users with better information tracking, user information pages can be switched by clicking the corresponding button on the leftmost navigation menu.



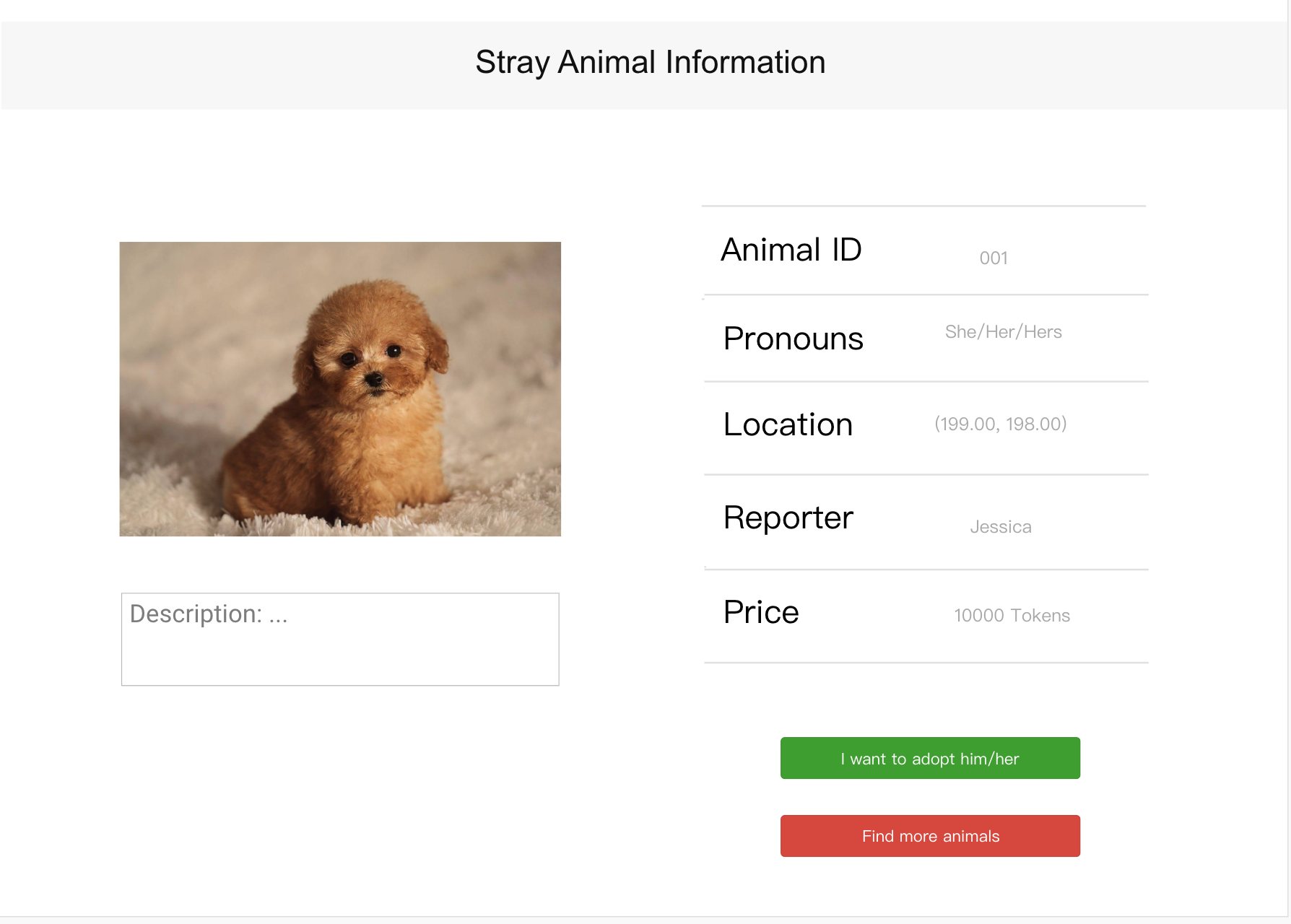
User profile page consists of a user’s personal information, including user’s username, ethereum account address, and the number of animals he has ever adopted and posted. Besides, the user can change his/her username, password, or log out to sign in using another account.



User transaction page is designed to collect all the transaction records of a user and display it in the form of an expandable table. The table entry in the main table shows the username of your seller/buyer and the date when the transaction happened. When you click the “+” button on the left of every table entry, a sub table appears and every entry of it records the information about the ether transaction with regards to the transaction in our site.

##### 3.1.4 Prototypes

Before we started the overall programming, we first created a prototype design for our Dapp. To build a low-fi prototype for the proposed animal adaptation Dapp front-end design, the software Mockingbot was utilized. The detailed prototype can be found in this link: <https://mockingbot.com/app/ndzotst3xvkmcgvfims31vq4xu5bk>. Note that there are many improvements after our first prototype, since some of the previous ideas don't fit into Solidity. For example, the price was originally designed in tokens units, but due to the fact that metamask doesn’t recognize the other ethereum account in this case, we switched it to Ether in our project programming. Therefore, some pre-designed page was abandoned as we programmed, but we follow the general design structure all the same.



#### 3.2 Back-end Design

Backend is designed to act like a traditional database, to retrieve information and store some defined states. But deploying backends by using ethereum makes our backend a safer and more trusted environment for the customers to handle their data.

##### 3.2.1 Design of storing data

1. The UserInfo data is to store an individual user data:

**struct** **UserInfo** {

string userName;

bytes32 passHash;

address accountAddress;

uint256 adoptedNum;

uint256 userNameIndex;

}

* userName: user name in plain text
* passHash: password after hashing, we do not store password as plain since it’s not safe
* accountAddress: the ethereum address of an account associate with this user name
* adoptedNum: how many animal has this user adopted

1. The AnimalInfo is to store an animal’s information posted by user:

**struct** **AnimalInfo** {

uint256 animalID;

string longitude;

string latitude;

string contactUserName;

uint64 price;

string imageBase64;

string title;

string description;

*// status = "MISSING", "FOUND"*

string status;

address **payable** seller;

string time;

string physicalAddress;

}

* animalID: an unique animal index number
* longitude: the longitude information of this animal’s position
* latitude: the latitude information of this animal’s position
* contactUserName: the user who post this animal information
* price: price to adopt this animal(in ether)
* imageBase64: animal image date(in base64 format)
* title: animal name
* description: a brief introduction of this animal
* status: the status of the animal(can be “MISSING” or “FOUND”)
* seller: the address of user who sell this animal
* time: the posted time of the animal information
* physicalAddress: the physical address of this animal

1. The transaction information of each user:

**struct** **TransactionInfo** {

address from;

address to;

string fromUser;

string toUser;

string time;

uint256 animalIndex;

string animalTitle;

uint64 animalPrice;

}

* from: the ethereum address of the user who make the payment
* to: the ethereum address of the user who receive the payment
* fromUser: the user name of the user who make the payment
* toUser: the user name of the user who receive the payment
* time: the time of this transaction
* animalTitle: the name of the animal
* animalPrice: how many ether has been transferred

1. users is to link user address to user information

*// user address to user information*

mapping (address => UserInfo) users;

1. activeUsers is to map to link user address to user uuid

*// user address to username*

mapping (address => bytes32) activeUsers;

1. transRecords is to link user address to user transaction information

*// user address to user related transaction information*

mapping (address => TransactionInfo[]) transRecords;

1. postAnimalRecords is to link user address to animal information

*// user address to user related posted animal information*

mapping (address => AnimalInfo[]) postAnimalRecords;

1. animalInfo is to store all the animal information

*// all animal information*

AnimalInfo[] animalInfos;

1. userName is to store all the user names

*// all existing user name*

string[] userNames;

##### 3.2.2 Design of events

Event is a method to send the transaction information that is needed by the frontend.

1. OperationEvents is to send out the information of whether the current transaction is successful and the corresponding message to the frontend, the eventType can be *"ANIMAL\_INFO\_OPS", "USER\_ACTIVE", "TRANSACTION", "REGISTRATION", "LOGOUT", "PASSWORD\_RESET", "USERNAME\_RESET"*

event OperationEvents(string eventType, string eventMsg, bool success);

1. LoginEvent is to return if the user has signed in the website and return the corresponding uuid to maintain an online status

event LoginEvent(bytes32 uuid, string eventMsg, bool success);

1. TransactionRecords is to return the transaction information depending on the user’s request

event TransactionRecords(TransactionInfo[] records, string eventMsg, bool success);

#### 3.3 Agent Design

In our project, Agent is designed to packaging all the backend APIs, and expose its own APIs to the frontend. This kind of design can make front-end code more compact. If the backend API has changed, we can just change the Agent’s internal implementations without changing the code of the frontend.

**class** **Agent** {

**constructor**(myAccount,uuid){

**this**.myAccount = myAccount;

**this**.uuid = uuid;

}

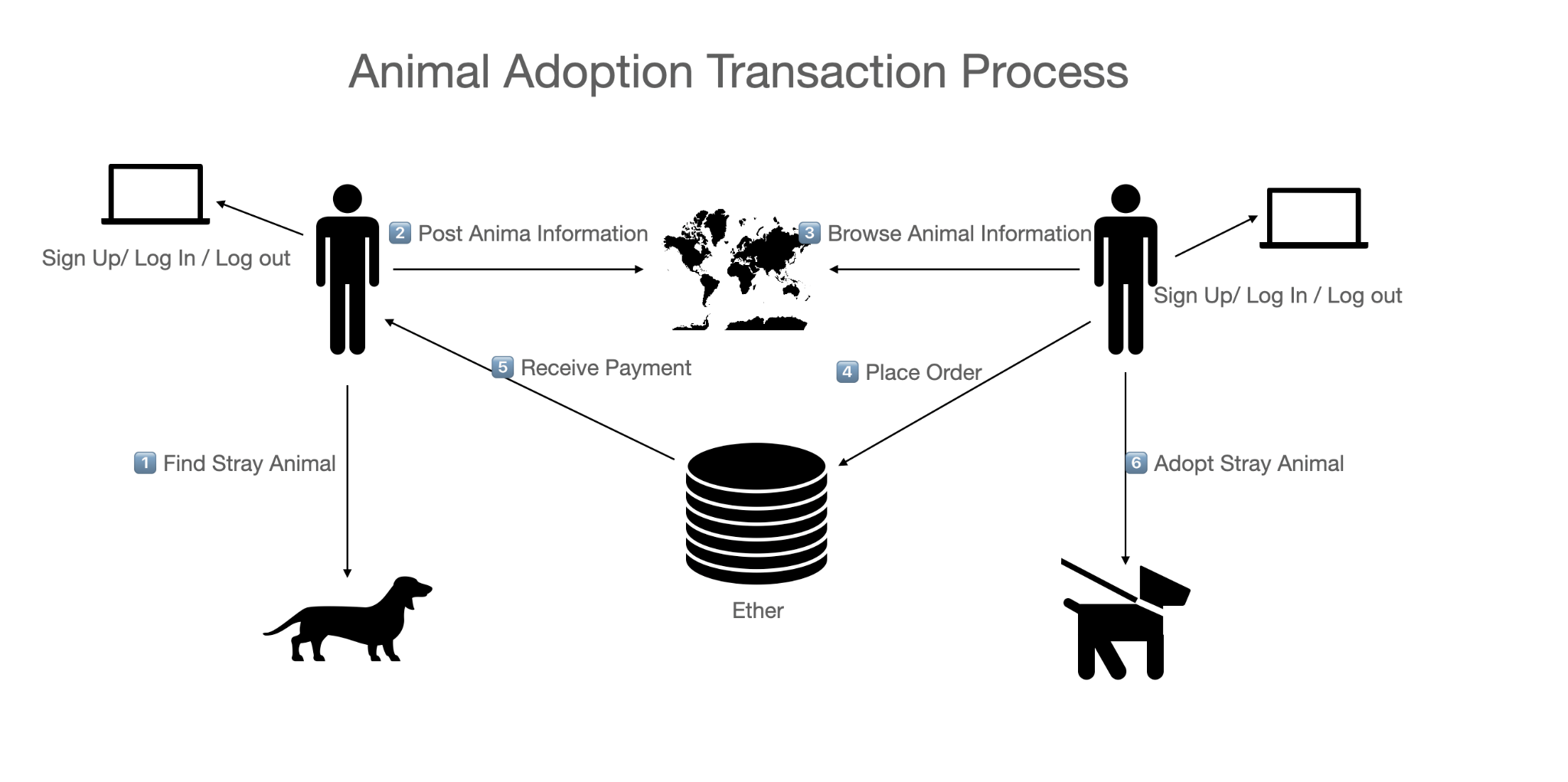
**Agent APIs...**}

#### 3.4 Overall Design

It is worth mentioning that the Agent plays an important role as the middleware in connecting the front-end and back-end and making the interaction between front and back end possible. Specifically, the user information, transaction information, posted animal information (including animal photos), were all stored in the back-end Solidity. While any information is required by the front-end, the Agent provides specific functions to retrieve those information and send them to the front-end.

#### 3.5 Use Cases Design

The overall animal adoption transaction process can be visualized as the figure below:



We divided all use cases into three groups based on its use case: users use case, post animal information use case, and transaction user case.

###### 3.5.1 User Use case:

To fulfill the functionalities of a common application, we enable the sign up, log in, log out functions as part of use case design. More detailed working flow is described below:

“Sign up -> Sign In -> Check User Profile Page -> Change Username -> Change Password -> Sign out -> Sign in with New Password and New Username”

###### 3.5.2 Post Animal Information Use case:

Users can post stray animals on the maps by providing required information, such as post title, time, location, price, photos, and description etc. More detailed working flow is described below:

“Sign in with Account1 -> Post An Animal Info -> Check the New Animal Info Marker -> Log out”

###### 3.5.3 Transaction Use Case:

There are two possible cases in the part of transaction: one is on the buyer side, the other is on the seller side. For buyers, they can first browse on the maps, and make a selection based on the pop up animal image. Then, they can check for detailed information and even adopt the stray animal if they are interested in it. After the order is confirmed successfully, the transaction history will be updated on both buyers and sellers sides with corresponding transaction information. More detailed working flow is described below:

“Buyer: Sign in with Account2 -> Check Animal Info Marker -> Click Adopt Button -> Confirm Order -> Check My Transaction History ->Log out”

“Seller: Sign in with Account1 -> Check My Transaction History -> Log out”

### Implementation

#### 4.1 Design Implementation

##### 4.1.1 Front-End

Using the React framework, we split the task of constructing the front end into implementing several React components based on our design in section 3.1. Besides, we use the ant design framework; the detailed API doc of it can be found [here](http://ant.design).

###### <App/>

App component is the entry of our front end. It defines routes which support routing all the other pages. For example, in the code below, if the url is “localhost:3000/signin”, React will render <SignInPage/> and display it in the user's browser.

<Route path="/signin">

<SignInPage />

</Route>

###### <Nav/>

<Nav/> the tab menu for navigation of our main page (see below).

<Nav>

<MapContainer name = "Animal Map"/>

<UserInfoPage name = "User"/>

</Nav>

It maps every of its child components into a tab item, and users will see the corresponding page rendered by the specific child component every time he/she clicks one of the tabs.

###### <CurrentLocation/>

<CurrentLocation/> helps the program get the user's current location via navigator.geolocation. When the current location is successfully achieved, local storage will be used to save position information so that it will be convenient for other components to get this information (see below). The CurrentLocation component also sets initial coordinates which could be displayed on the map when current location can not be taken through Google Map API.

**componentDidMount**() {

**if** (**this**.props.centerAroundCurrentLocation) {

**if** (navigator **&&** navigator.geolocation) {

navigator.geolocation.**getCurrentPosition**(pos **=>** {

const coords = pos.coords;

localStorage.**setItem**('lat',coords.latitude);

localStorage.**setItem**('lng',coords.longitude)

**this**.**setState**({

currentLocation: {

lat: coords.latitude,

lng: coords.longitude

}

});

});

}

}

**this**.**loadMap**();

}

###### <MapContainer/>

<MapContainer/> implements the basic map features and markers with infowindow pop up. All the animal information and current location are shown as different markers in the map. To achieve this, a marker list is created to store information including a Base64 image. And the list of animals maps the marker component. The markers are iterated to build different infowindows shown as below.

{**this**.state.animals.**map**((animal, index) **=>** (

<Marker

key={index} // *Need to be unique*

onClick={**this**.onMarkerClick}

name={animal.title}

position={animal.position}

id = {`animal\_marker\_${index - 1}`}

imageBase64 = {animal.imageBase64}

>

</Marker>

))}

<InfoWindow

marker={**this**.state.activeMarker}

visible={**this**.state.showingInfoWindow}

onClose={**this**.onClose}

>

<div align="center">

{

**this**.state.activeMarker **!=** **null** **&&** **this**.state.activeMarker.name **!=** "Post Animal" **?**

<img width="80" height="80" alt="star" src={**this**.state.activeMarker.imageBase64}/> **:** ""

}

</div>

<div align="center">

{

**this**.state.activeMarker **!=** **null** **&&** **this**.state.activeMarker.name **!=** "Post Animal" **?**

<a href={'/animalinfo/'+**this**.state.activeMarker.id}><font size="3">{**this**.state.activeMarker.name}</font></a> **:** ""

}

</div>

{

**this**.state.activeMarker **!=** **null** **&&** **this**.state.activeMarker.name **==** "Post Animal" **?**

<a href={'/post'}><font size="5">{**this**.state.activeMarker.name}</font></a> **:** ""

}

</InfoWindow>

</CurrentLocation>

</div>

###### <SignInPage/>

<SignInPage/> implements the login page. It returns a form with a button which users can click to submit a login request with his/her username and password he/she just inputs,and then pop up messages to notify users if this login in behaviour is successful or not. This page also contains a link to sign up page, in case a new user wants to join our community. The onClick function logIn() is as below.

async logIn(){

this.state.myAgent.initialize();

var input\_usernmame = document.getElementById("username").value;

var input\_password = document.getElementById("password").value;

var logInfo = await this.state.myAgent.login(input\_usernmame,input\_password);

if(logInfo[0]){

this.setState({

signedin : logInfo[0]

})

}

else{

message.error(logInfo[1]);

}

localStorage.setItem(this.state.myAgent.myAccount, logInfo[2]);

}

###### <SignUpPage/>

<SignInPage/> implements the registration page. It functions similarly : it returns a form with a button which users can click to submit a registration request with a new unique username and password he/she just inputs, and pop up messages to notify users if this login in behaviour is successful or not. It can also check if the username is taken by another user, if your password does not meet the requirement, and if you confirm your password in the third input.

checkPwdConfirmed(){

var input\_password = document.getElementById("new\_pwd").value;

var input\_confirmed = document.getElementById("confirm\_pwd").value;

this.setState({

pwd\_confirmed : input\_password.length > 0 && input\_confirmed.length > 0 ? (input\_confirmed == input\_password ? 1 : -1) : 0

})

}

This page also contains a link to <SignInPage/> in case a new user already has an account. The onClick function handleSignUp() is as below.

async handleSignUp(){

var input\_usernmame = document.getElementById("username\_signup").value;

var input\_password = document.getElementById("password\_signup").value;

var myAgent = new Agent(null,null)

await myAgent.initialize()

let callback = await myAgent.registeration(input\_usernmame, input\_password);

this.setState({

signedup : callback[0],

isModalVisible : true,

fail\_msg: callback[1]

})

return;

}

###### <TabMenu/>

<TabMenu/> component functions in the same way as <Nav/>, except that this tab menu has a vertical layout.

###### <UserInfoPage/>

<UserInfoPage/> is the entry to the whole user information reviewing section. The only functionality of it is to encapsulate <UserProfilePage/> and <UserTransPage/> using <TabMenu/>

###### <UserProfilePage/>

<UserProfilePage/> collects user information by calling interfaces provided by the agent and displays them on the page. Besides, it also has three buttons: reset username, reset password, and log out. After clicking the first two button, a modal will pop up and allow user to input a new username/password

<Modal

title = "Reset username"

width = {600}

visible = {this.state.settingUname}

onOk = {()=>this.updateUsername()}

okText = "Reset"

onCancel = {()=>this.cancelChangeUname()}

destroyOnClose

>

<Form

style={{padding:0}}

>

<Form.Item

label="New Username"

hasFeedBack

validateStatus = {this.state.unique\_name == -1 ? "error" : "success"}

help = {this.state.unique\_name == -1 ? "Your name has been taken! Chooese another one" : " " }

style = {{textAlign:"left"}}

>

<Input placeholder = "new username" id = "new\_username" style = {{width:400}} />

</Form.Item>

</Form>

</Modal>

logout button will call myAgent.logout() and pop up messages to notify users if the logout operation is successful.

async logOut(){

this.state.myAgent.initialize();

this.state.myAgent.uuid = localStorage.getItem(this.state.myAgent.myAccount);

var logOutInfo = await this.state.myAgent.logout();

if(logOutInfo[0]){

message.success(logOutInfo[1])

this.setState({

loggedOut : true

})

}else{

message.error(logOutInfo[1])

this.setState({

logOutModalVisible : false

})

}

}

###### <UserTransPage/>

<UserProfilePage/> collects user transaction records by calling interfaces provided by the agent and displays them on the page in the form of an expandable table with pagination. To implement that, we set the pagination attribute in <Table/> and provide a function to map every main table entry to its sub table.

expandedRowRender = (record,index) => {

const columns = [

{title: 'From', dataIndex: 'from', width:50},

{title: 'To', dataIndex: 'to',width:50}

];

return <Table columns = {columns} dataSource = {[record]} pagination = {false}/>

}

<Table

dataSource = {this.state.records}

columns = {columns}

pagination = {

{

position : ['bottomCenter'],

total : this.state.records.length,

pageSize : 3,

responsive : false,

showTotal : total => `Total ${total} items`

}

}

expandable = {{ expandedRowRender: this.expandedRowRender}}

/>

###### <PostInfoPage/>

<PostInfoPage/> renders a form for users to input animal information. We use <DatePicker/> in ant design allowing users to efficiently provide a time and transform it into a string using javascript moment package.

disabledDate(current){

return current > moment().endOf('day');

}

<DatePicker

id = "date"

showTime

disabledDate = {this.disabledDate}

style = {{width:300}}

/>

We set the user's current location(latitude and longitude) as the default value of the location input.

<Form.Item label = "longitude" rules={[{ required: true, message: 'Please enter a longitude!' }]} >

<Input id = "longitude" placeholder = "longitude" defaultValue = {lng}></Input>

</Form.Item>

<Form.Item label = "latitude" rules={[{ required: true, message: 'Please enter a latitude!' }]} >

<Input id = "latitude" placeholder = "latitude" defaultValue = {lat}></Input>

</Form.Item>

We also implement a text area where users can freely give any information they want as it can be resized easily.

<Input.TextArea id = "description"

placeholder = "Tell us more about this little thing~"

allowClear

showCount

maxLength = {200}

/>

###### <AnimalInfoPage/>

<AnimalinfoPage/> returns a page which contains all the information about the picked stray animal, including animal photos, id, title, position in terms of longitude and latitude, address, reporter, price, and description provided by the reporter.

###### <OrderConfirmPage/>

<OrderConfirmPage/> is the next page of <AnimalInfoPage/> if the user is interested in this option and would like to place an order. The purpose of this page is to help users double check their order and personal information. <OrderConfirmPage/> is delivered as a combination of animal information and personal information confirmation. Where animal information is almost the same as the previous page except for omitting animal images. Regarding the personal information confirmation section, it includes users username and account address.

##### 4.1.2 BackEnd

1. **function getAdoptedNum(bytes32 uuid) public view returns(uint256)**

Get the adoption number that is stored in UserInfo.

1. **function resetUserName(string memory newName, bytes32 uuid) public** returns(bool)

Change the userName that is stored in UserInfo

1. **function getMyBalance(bytes32 uuid) public returns(bool, string memory, uint256)**

Get the account balance of the msg.sender.

1. **function getTransRecords(bytes32 uuid) public returns(TransactionInfo[] memory, string memory, bool, uint256)**

Get all the transactionRecords that are associated with the msg.sender.

1. **function getPostedAnimal(bytes32 uuid) public returns(AnimalInfo[] memory, string memory, bool, uint256)**

Get all the posted animal records that are associated with the msg.sender.

1. **function resetPassword(string memory \_old\_password, string memory \_new\_password, bytes32 uuid) public returns(bool)**

Change the password from UserInfo of the current user.

1. **function logout(bytes32 uuid) public**

Logout current user.

1. **function getAnimalNearBy(int64 top, int64 bottom, int64 left, int64 right, int64 radius, bytes32 uuid) public returns(AnimalInfo[] memory, uint256, bool)**

Get nearby posted animal information based on current geographical location information.

1. **function adoptAnimal(uint256 \_index, string memory \_time, bytes32 uuid) public payable returns(bool)**

Adopt the selected animal, make the payment to smart contract and transfer

the money to the seller account. Also this function will record the transaction

information and change the status in the animal info.

1. **function postAnimalInfo(string memory \_longitude, string memory \_latitude, uint64 \_price, string memory \_imageBase64, string memory \_title, string memory \_description, string memory \_time, string memory \_physicalAddress, bytes32 uuid) public returns(bool)**

User posted animal information to the smart contract. The smart contract will store a new AnimalInfo struct.

1. **function getUserName(bytes32 uuid) public returns(bool, string memory, address)**

Get the current username associate with msg.sender.

1. **function register(string memory \_userName, string memory \_password) public returns(bool)**

Register a new user account with an ethereum account address - msg.sender.

The smart contract will store data in UserInfo.

1. **function login(string memory userName, string memory password, string memory timestamp) public returns(bool)**

Use this function to login users that associate with the current ethereum

account address. This function will return an uuid which is a unique number to

maintain user’s login status.

1. **function checkUUID(address \_accountAddr, bytes32 uuid) public view returns(bool)**

In most functions that require the user to be in login status, this function will

be called to make sure that the user who sent the request has login.

1. **function compareStrings(string memory a, string memory b) public pure returns (bool)**

This is a helper function to compare if two strings are equal in the solidity.

1. **function hash(string memory str) public pure returns(bytes32)**

This is a helper function to generate the hash value of a string value.

##### 4.1.3 Agent

1. **async getWeb3Provider()**

Get the web3 provider, we use MetaMask as default method.

1. **async initialize()**

Initialize the Agent: connect to web3 provider and getting the current account

address.

1. **async isUniqueName(userName)**

Check if the name to be registered is unique in the bankend data.

1. **async getAdoptedNum()**

Get the adopted animal number of the current user.

1. **async getUserName()**

Get current user name.

1. **async getTransRecords()**

Get all transaction records of the current user.

1. **async getPostedAnimalRecords()**

Get all posted animal records.

1. **async resetUserName(newUsername)**

Change the current user’s username.

1. **async resetPassword(old\_password, new\_password)**

Change the current user’s password.

1. **async registration(username, password)**

Register a new user with a current ethereum account address.

1. **async login(username, password)**

Login an user with current ethereum account address.

1. **async logout()**

Logout the current user.

1. **async getBalanceOf()**

Get the balance of the current user.

1. **async postAnimal(longitude, latitude, currentTime, price, imageBase64, title, description, physicalAddress)**

Post one animal information with the current user.

1. **async getAnimalNearBy()**

Get nearby animal information posted by all the users.

1. **async adoptAnimal(index, price)**

Adopt an animal with the current user.

#### 4.2 Problems Encountered and Relative Solution

1. In <PostInfoPage/>, we need to allow users to upload images, but as it is a decentralized app, there are no databases for us to store data in. After careful consideration, we decided to transform an image into base64 format so that it can be directly processed by javascript. The transform process is as below.

const imgChange = e => {

    var fileList = e.target.files;

    var file\_num = fileList.length;

    var AllowImgFileSize = 10240;

    for(let i = 0; i < fileList.length;i++){

        let reader = new FileReader();

        reader.readAsDataURL(fileList[i]);

        reader.onload = function(e){

            if (AllowImgFileSize != 0 && AllowImgFileSize < reader.result.length) {

                message.error("file size exceeds 10kB!");

                return;

            }else{

                console.log(reader.result)

                imgUrlBase64.push(reader.result);

            }

            return;

        }

    }

    message.success(`${file\_num} file${file\_num > 1 ? "s" : ""} ha${file\_num > 1 ? "ve" : "s"} been successfully uploaded`);

    console.log(imgUrlBase64);

}

For every file the user uploads, we create a fileReader and call fileReader.readAsDataURL, then we implement the onload function which is called every time after the reader processes a file. In this way, image files become a variable in our programs.

2. Google-maps-react library defines that changes of InfoWindow influence all the markers on the map. However, for <MapContainer/> in the project, InfoWindows of current location and animal information are designed to be presented in different ways. To solve the problem, first the marker representing the current location was given a certain name called ”Post Animal” since the program is designed to post animals by clicking current location InfoWindow. Then, we add conditional statements to check. If the name equals “Post Animal”, there will be a link to the post animal page. If not, an animal image and a link to the animal information page will be displayed in InfoWindows (see below).

<InfoWindow

marker={this.state.activeMarker}

visible={this.state.showingInfoWindow}

onClose={this.onClose}

>

<div align="center">

{

this.state.activeMarker != null && this.state.activeMarker.name != "Post Animal" ?

<img width="80" height="80" alt="star" src={this.state.activeMarker.imageBase64}/> : ""

}

</div>

<div align="center">

{

this.state.activeMarker != null && this.state.activeMarker.name != "Post Animal" ?

<a href={'/animalinfo/'+this.state.activeMarker.id}><font size="3">{this.state.activeMarker.name}</font></a> : ""

}

</div>

{

this.state.activeMarker != null && this.state.activeMarker.name == "Post Animal" ?

<a href={'/post'}><font size="5">{this.state.activeMarker.name}</font></a> : ""

}

</InfoWindow>

3. In the transaction process, we initially considered using tokens instead of ethers, and we designed the add token process in the <AddTokensPage/>. The design point here is that if we want to spend tokens, we first need to approve enough tokens. However, during actually coding process, we realized that there exists some issues with metamask, since it regarded all other ethereum account invaid, by returning “not valid ethereum accounts” except for the account it currently selected. Therefore, we ended up switching the transaction from using tokens to ethers and re-programming back-end design. And the original <AddTokenPage/> was omitted, as we mentioned in the prototypes design part.

### Testing

#### 5.1 Front-end Testing

We have conducted manual testing according to use cases designed on Section 3.5.

#### 5.2 Back-end Testing

Backend testing is mainly done by using “Truffle test”. There are a total of 15 test cases, each case is targeted to test one particular function of the solidity smart contract.

Test cases Summary:

1. **The deployment should be done successfully**

Test if the smart contract is deployed successfully.

1. **Register user1**

Register user1 with some user information.

1. **Login user1**

Sign in user1 with the username and password defined in test case2.

1. **User1 post one animal information**

User1 posts one self-defined animal information to the smart contract.

1. **User1 get nearby animal information**

User1 gets nearby animal information, which should get the information user1 just posted.

1. **Test user1 reset password**

Reset the password of User1.

1. **Test user1 reset username**

Change the username of User1.

1. **Test user1 logout**

Logout user1.

1. **Login user1**

Login user1 with the new username and password.

1. **Register User2**

Register user2 with some user information.

1. **Login user2**

Sign in user2 with the username and password defined in test case10.

1. **User2 adopt user1 posted animal**

User2 adopts the animal that was posted by user1. The status of the animal should change in AnimalInfo struct. There should be a payment made by User2 to smart contract and smart contract to User1.

1. **Test get transaction records**

Get all transaction records of the current user.

1. **Test user1 logout**

Logout user1.

1. **Test user2 logout**

Logout user2.

The testing result is belowed:

Contract: AdoptionCentre

✓ The deployment should be done successfully

✓ Register user1 (211ms)

✓ Login user1 (79ms)

✓ User1 post one animal information (176ms)

✓ User1 get nearby animal information (138ms)

✓ Test user1 reset password (175ms)

✓ Test user1 reset username (100ms)

✓ Test user1 logout (85ms)

✓ Login user1 (45ms)

✓ Register User2 (126ms)

✓ Login user2 (49ms)

✓ User2 adopt user1 posted animal (177ms)

✓ Test get transaction records (127ms)

✓ Test user1 logout (63ms)

✓ Test user2 logout (70ms)

15 passing (2s)

### Discussion & Future Improvement

Due to the time limit, our main goal in this term project is to build a Truffle + React + Web3js development environments application in stray animals adoption. As we programmed this project, we realized that there still exist several technique challenges that were tough to solve in a limited time. Specifically, for animal images upload part, only images smaller than 4kb could be processed successfully because of the gas limit in Solidity. To make current Dapp more user-friendly, we anticipate adding CSS properties to every component in the front end so that they can look more organized. We also expect to provide a location search bar for users so that they can input a street name and front end will automatically translate it into their latitude and longitude. Adding a process in the back end so that a user who is inactive for a long time will automatically log out could also make this Dapp more usable. Also, automatically refreshing the page every 30 sec or even less will manage a large users group and their transaction synchronously.

While building the smart contract, we found out that we cannot store too much information into one smart contract - since it will create a deployment issue which is caused by the gas limit. We are thinking about using multiple smart contracts to store the data separately, which will avoid the deployment constraint and increase the scalability of our system.

One significant limitation exists in current decentralized applications is the lack of communication between buyer and sellers. Unlike most common online transaction platforms, where customers can freely chat with sellers to inquire about any concerns about the products or even bargain, users in the current version Dapp can only leave their contact information in the animal descriptions for further communication. Enabling a chat tool either inside or attached to the Dapp would be definitely a major next step.

Regarding functionalities improvements, taking animal adoption non-profits organizations as another user group into consideration would greatly enlarge the target user groups, boost the communication between single users and organizations and increase the efficiency of helping stray animals find their new home. It needs to be mentioned that if animal adoption organizations are included as potential users, it is also essential to take security issues, such as identification verification, into consideration.

### Conclusion

We have presented an animal adoption decentralized application, Animadopt, for assisting stray animal adoption communities to care for homeless animals using React.js with Javascript as the main front-end language and Solidity as the back-end programming language. The application enables users to post, share and adopt the stray animals on the Google Maps APIs in the form of markers and secures the transaction as users in Dapps can transact directly with each other using cryptocurrencies without third-party disturbing. We also discuss development directions for the future. This work provides a relatively convenient tool for pet-lovers to put their joint efforts to help stray animals find their new home.

### *Acknowledgment*

We thank Professor Zhehua Wang for his feedback and suggestions during this work. Also, we are grateful to the Solidity, React, Javascript, CSS open source communities for assembling some of the great software tools we have used here.

### 

### References

1. Canada Animal Shelter Statistics Report <https://humanecanada.ca/wp-content/uploads/2020/11/Humane_Canada_Animal_shelter_statistics_2019.pdf>
2. Edoption – Pet Adoption App Concept. <https://project365.design/2018/04/08/day-98-edoption-pet-adoption-app-concept/>
3. Kolandaisamy, R., Subaramaniam, K., Kolandaisamy, I., & Siew Li, L. (2016). Stray animal mobile app. In Conference: Regional Conference On Sciences, Technology and Social Science (RCSTSS). Retrieved from <https://www.researchgate.net/publication/312057960_Stray_Animal_Mobile_App>
4. Animal-Free: https://www.animalfree.info/en
5. L.Conway. “Blockchain Explained” Investopedia. <https://www.investopedia.com/terms/b/blockchain.asp> (accessed April. 12, 2021)
6. Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." *International Journal of Web and Grid Services* 14.4 (2018): 352-375.

### 