COMP9900 Information Technology Project

T3, 2020

Online Property Sales

Project Report

Group Name: COMP9900-W15B-FVC5

Name	**Email**	**Student ID**	**Role**
Anqige Wu	z5199351@ad.unsw.edu.au	z5199351	Developer/Master
Yuchen Yang	z5189310@ad.unsw.edu.au	z5189310	Developer
Rong Zhen	z5225226@ad.unsw.edu.au	z5225226	Developer
Dan Su	z5226694@ad.unsw.edu.au	z5226694	Developer
Jiaqi Sun	z5233100@ad.unsw.edu.au	z5233100	Developer

Submission Date: 16/11/2020

Table of Contents

1. Overview

1.1 Background

With the development of information technology, people prefer to deal with their daily routines online. Especially in the face of COVID-19, many different types of online services have achieved success, including online office, online learning, and online shopping. These online services provide opportunities for people to complete different missions remotely, as well as improve their life quality. The online economy is driving social development and making human life more and more convenient.

Recently, online real estate transactions have also developed rapidly to fit the customers' requirements. On these platforms, sellers are able to attract more bidders to participate in the auctions by posting detailed property information online. Meanwhile, buyers do not need to spend a lot of time on-site inspections and they can compare different properties online easily. There is no need to keep an eye on price trends on the auction site, thus saving time and transportation costs. Online auctions have made great achievements in the trading of second-hand items, such as eBay, but for properties and expensive artworks, the users are more special and targeted, which forces the platforms to consider more factors to give the customers the best experience.

Nowadays, some existing platforms which can provide the property auction service for online users have been developed, but most of them use Agents to help sellers sell properties, which makes the process cumbersome and inconvenient for the seller to some extent.

Our project aims to provide users with an online platform for property auctions, at which the sellers can independently upload the property information after being certified (i.e. verify the real estate certificate), and the bidder can bid the properties if they are interested in.

1.2 System Design and Software Architecture

The entire system of our project consists of four parts: user type, front-end layer, back-end layer and data layer. Our users are mainly divided into 4 types, the Front-end is mainly written in JavaScript language, the Back-end is mainly written in Python3 language, and the database mainly uses SQLite database.

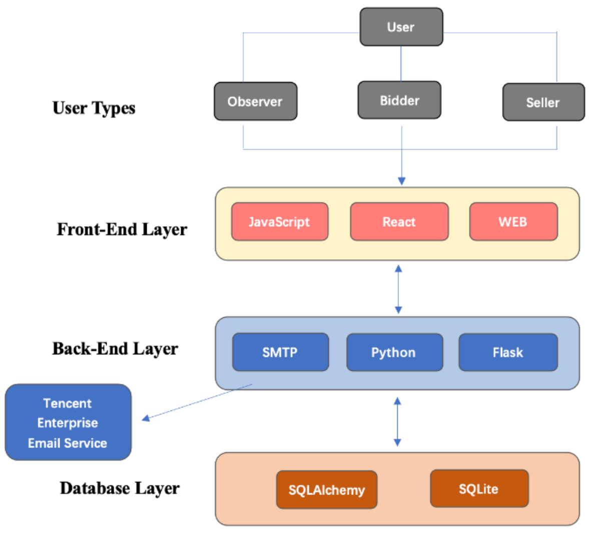


Figure 1. Software architecture layers

1.2.1 User type

Our ‘Online Property Sales’ project is aiming to provide an online property auction platform for users with such needs. There are four different types of potential users, including users, sellers, observers, and bidders, and the relationship is illustrated in Figure 1. For different types of users, we provide different services.

For a user who is not logged in, he can perform basic browsing and search operations on the website, a common user can view the auction details, including property information and current bid, however, he cannot choose to become a seller, bidder or observer if not logged in. Besides, a user who is not logged in cannot see the bid history of each auction.

Once a user logs in, he can choose to become a seller and post his property on the website for sale.

When posting a new property, the seller needs to fill in the basic information, including the reserve price, the size of the house, the address, the number of bedrooms, bathrooms, and garages. In addition, he needs to upload the property ownership certificate for the website manager to ensure the authenticity and security of the transaction.

The reserve price is not shown to bidders and observers. It is only used for the website to judge if the auction is successful. Once the auction time is over, if the final bid price, namely the current price is lower than the reserve price, then the auction failed. On the contrary, if the bid price is higher than the reserve price, then the auction is a success. If an auction is successful, both seller and the highest bidder will receive an email including each other’s contact details. If an auction fails, only the seller will receive an email including the highest bid price.

Besides, the user can choose to become a bidder. Once a bidder passed the real-name authentication verification and has uploaded the relevant payment information, he can participate in the real estate auction that he is interested in.

A bidder can not only view his last bid and current bid, but also the bid history of the auction he is involved in.

Finally, the user can also choose to become an observer. An observer can view the bid history of the property on the auction details page. If an observer is not interested in the auction anymore, he can choose to cancel the observation.

For these four user types, some recommendations based on their search history will be given for further.

1.2.2 Front-end layer

The front-end is mainly developed by JavaScript and React is used for connection between UI and the back-end. Our front-end is used in a web application, which is able to collect the inputs of the user layer and send all information to the back-end layer.

In this project, we designed different components showing our website features, including searchBox, auctionCard, Modals, Alerts, etc., and assembled them into different pages. React Bootstrap and Ant Design of React are used in this layer to enrich and beautify the web application.

1.2.3 Back-end layer

Our back-end is mainly built using Python and Flask. We obtain the information entered by the user from the front-end through the request function, and define whether the information meets our requirements and store it in our SQLite relational database table at the back-end, and finally return the json data as result.

We use SQLAlchemy to search for the data we need in relational database tables and map the table structure of the relational database of the relational database to the session object used in the back-end.

In addition, for user information verification and security, we use a hash function in Werkzeug security. When the user logs in, we hash the password entered by the user. At the same time, we will also grab the password of the user account from the database and encode it with a hash function. When the hashed codes of the two passwords are the same, it proves that the user's password is correct and the user login is successful.

Finally, we also import SMTPlib which simply encapsulate the SMTP protocol, and then sent related notification emails.

1.2.4 Database Layer

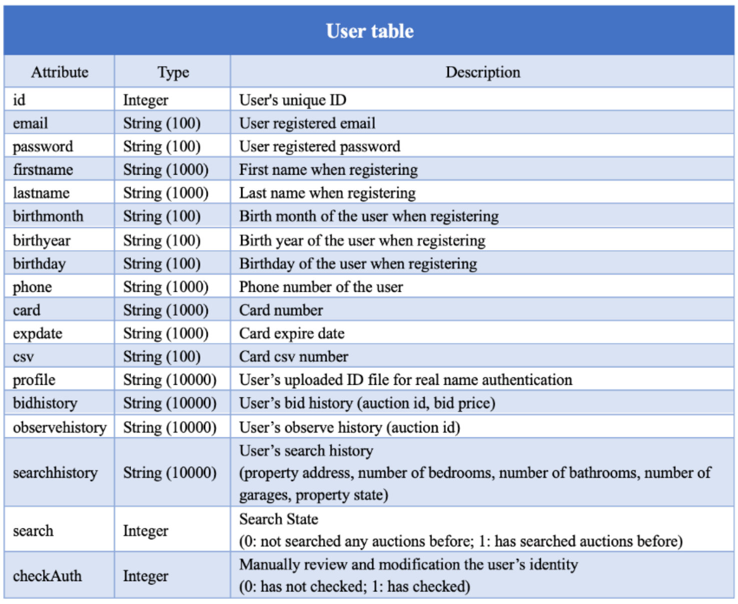
We use SQLite as our database and SQLAlchemy to operate database efficiently and with high performance.

The database mainly has two tables, user table and auction table. This first table is user table that stores information about users including their personal information, payment details, bid history, observe history and user’s search history. Bid history mainly stores auction ID and bid price. Observe history mainly stores auction id for viewing real estate. Search history mainly stores the records the user searched for real estate. And we set a search status to check whether the user searched before. Through search history records, we can recommend real estate to users.

The second table is auction table that stores auction property detail, id and bid price of bidder participating in the auction and observer email.

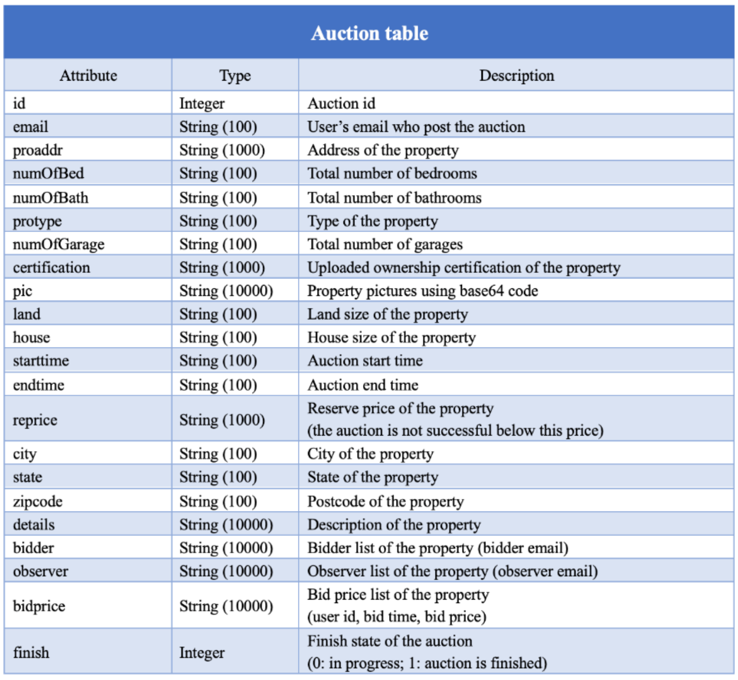
It is worth mentioning that we have made a detailed classification of the status of each property and marked it with the information of finish. When finish equals to 0, it means the status is waiting for the transaction, and when finish equals to 1, it means that the real estate transaction has finished.

Table 1. Database – User table



User table		
Attribute	Type	Description
id	Integer	User's unique ID
email	String (100)	User registered email
password	String (100)	User registered password
firstname	String (1000)	First name when registering
lastname	String (1000)	Last name when registering
birthmonth	String (100)	Birth month of the user when registering
birthyear	String (100)	Birth year of the user when registering
birthday	String (100)	Birthday of the user when registering
phone	String (1000)	Phone number of the user
card	String (1000)	Card number
expdate	String (1000)	Card expire date
csv	String (100)	Card csv number
profile	String (10000)	User's uploaded ID file for real name authentication
bidhistory	String (10000)	User's bid history (auction id, bid price)
observehistory	String (10000)	User's observe history (auction id)
searchhistory	String (10000)	User's search history (property address, number of bedrooms, number of bathrooms, number of garages, property state)
search	Integer	Search State (0: not searched any auctions before; 1: has searched auctions before)
checkAuth	Integer	Manually review and modification the user's identity (0: has not checked; 1: has checked)

Table 2. Database – Auction table



Auction table		
Attribute	Type	Description
id	Integer	Auction id
email	String (100)	User's email who post the auction
proaddr	String (1000)	Address of the property
numOfBed	String (100)	Total number of bedrooms
numOfBath	String (100)	Total number of bathrooms
protype	String (100)	Type of the property
numOfGarage	String (100)	Total number of garages
certification	String (1000)	Uploaded ownership certification of the property
pic	String (10000)	Property pictures using base64 code
land	String (100)	Land size of the property
house	String (100)	House size of the property
starttime	String (100)	Auction start time
endtime	String (100)	Auction end time
reprice	String (1000)	Reserve price of the property (the auction is not successful below this price)
city	String (100)	City of the property
state	String (100)	State of the property
zipcode	String (100)	Postcode of the property
details	String (10000)	Description of the property
bidder	String (10000)	Bidder list of the property (bidder email)
observer	String (10000)	Observer list of the property (observer email)
bidprice	String (10000)	Bid price list of the property (user id, bid time, bid price)
finish	Integer	Finish state of the auction (0: in progress; 1: auction is finished)

2. Function Description and Corresponding Objectives

2.1 Homepage

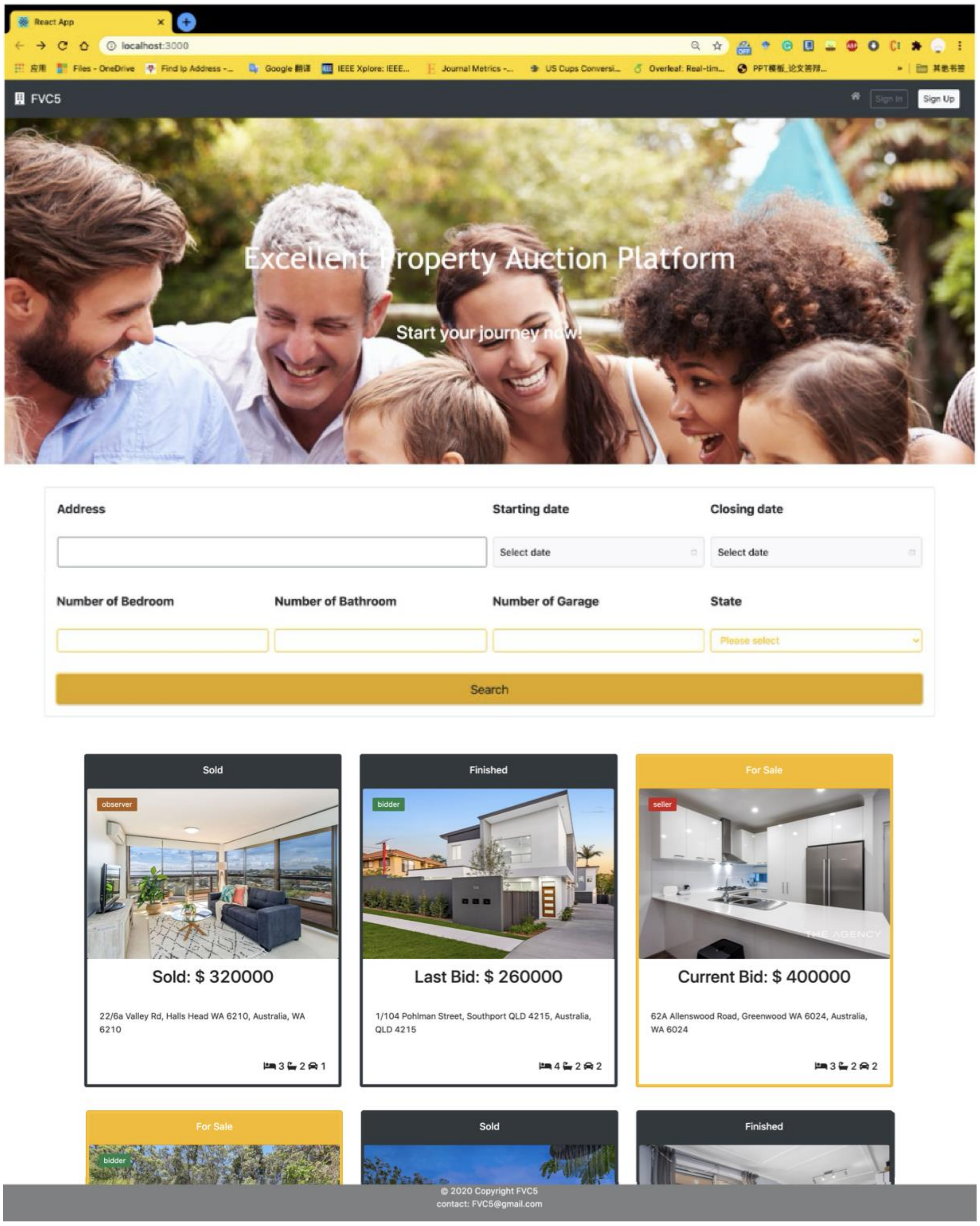


Figure 2. The homepage layout

FVC5 website provides an auction platform for selling and bidding properties online without agents. The homepage of this website is shown in Figure 2.

2.1.1 Navbar

The Navbar before login includes four buttons. By clicking the logo on the top left or the house-shaped icon on the top right, the user can jump to the homepage from any pages. The '*Sign In*' and '*Sign Up*' buttons on the right of the Navbar will be described in section 2.2.

2.1.2 Search Box

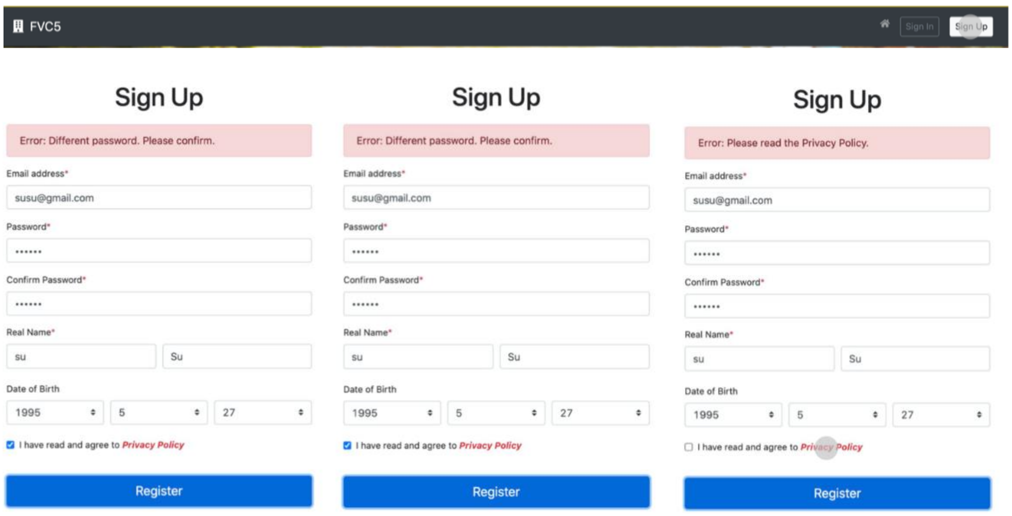
This website allows visitors (including unregistered users) to browse all the properties uploaded on the FVC5 website by clicking the '*Search*' button or enter some interested features (address, auction start/end time, bedroom/bathroom/garage number and state) to get the filtered auctions.

The auction cards in different auction states are shown in different colors: '*Sold*' (successfully sold) and '*Finished*' (not sold) shown in black border; '*For Sale*' (in progress) shown in yellow border. Clicking the auction cards will jump to the auction details page, which will be described in section 2.5.

The auction cards also show the user types by different colors of tags at the top left corner.

2.2 Register, Log-in and complete Profile

2.2.1 Register



Sign Up

Error: Different password. Please confirm.

Email address*: susu@gmail.com
Password*: ••••••
Confirm Password*: ••••••
Real Name*: su | Su
Date of Birth: 1995 | 5 | 27
[✓] I have read and agree to *Privacy Policy*

Sign Up

Error: Different password. Please confirm.

Email address*: susu@gmail.com
Password*: ••••••
Confirm Password*: ••••••
Real Name*: su | Su
Date of Birth: 1995 | 5 | 27
[✓] I have read and agree to *Privacy Policy*

Sign Up

Error: Please read the Privacy Policy.

Email address*: susu@gmail.com
Password*: ••••••
Confirm Password*: ••••••
Real Name*: su | Su
Date of Birth: 1995 | 5 | 27
[ ] I have read and agree to *Privacy Policy*

Figure 3. The error alert messages on the registration page

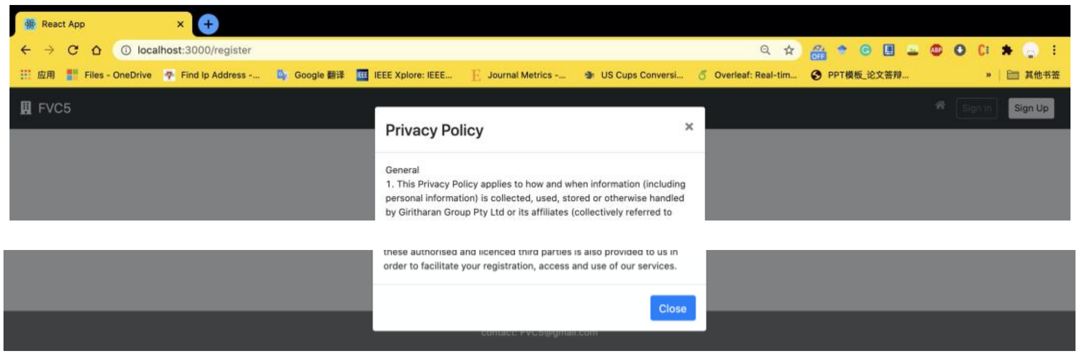
By clicking the '*Sign Up*' button or clicking the sentence '*Do not have an account? Register!*' in '*signin*' page, the user will go to the '*register*' page (Figure 3). On this page, the user can register a personal account by using the Email address and set a password for signing in to get more services on this website. In order to protecting users' privacy, the system will use SHA256 to encrypt the password.

There are three situations where the page will throw an alert message when the user wants to register an account (Figure 3):

a. '*Password*' and '*Confirm Password*' fields are different.

b. Some of the required fields are left empty.

c. The checkbox is not checked.

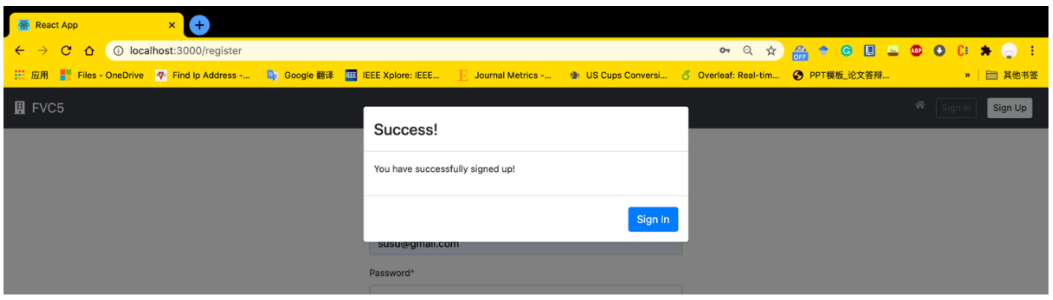


General
1. This Privacy Policy applies to how and when information (including personal information) is collected, used, stored or otherwise handled by Giritharan Group Pty Ltd or its affiliates (collectively referred to

these authorised and licenced third parties is also provided to us in order to facilitate your registration, access and use of our services.

Figure 4. The Privacy Policy Modal

If a user wants to check the content of the Privacy Policy, he can click the red words '*I have read and agree to Privacy Policy*' in the '*register*' page, and then a modal will show up and can be folded by the '*Close*' button (Figure 4).

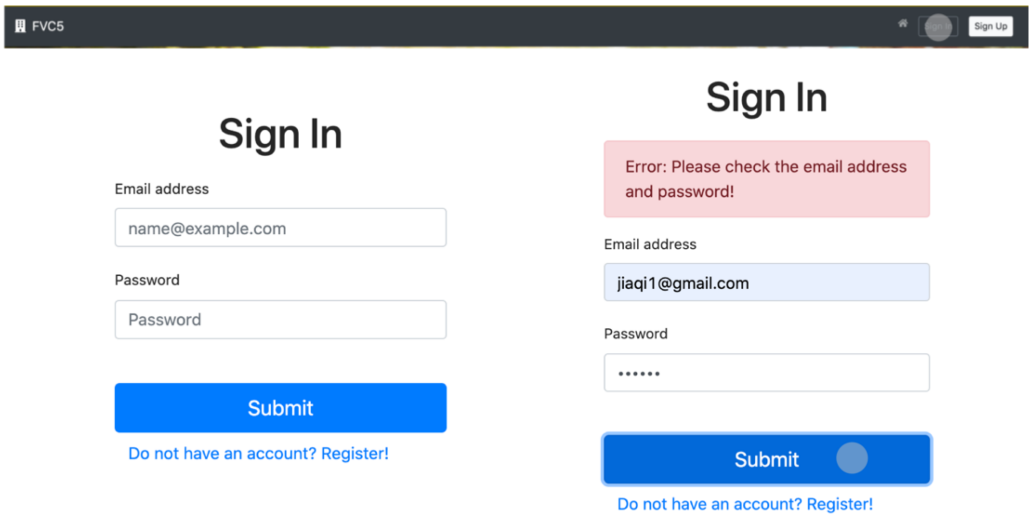


You have successfully signed up!

Figure 5. The modal for successful registration

The modal component can take the user to '*signin*' page by clicking the '*Sign In*' button on the modal after registering an account successfully (Figure 5).

2.2.2 Sign In


FVC5
Sign Up
Sign In
Email address
Password
Submit
Do not have an account? Register!
Sign In

Error: Please check the email address and password!

Email address: jiaqi1@gmail.com

Password: ••••••

Submit
Do not have an account? Register!

Figure 6. The sign-in page and error alert message

Clicking the ‘*Sign In*’ button on the Navbar will jump to the ‘*signin*’ page. The user can log in by using the previously registered Email address and Password to get permission to use further functions. If the Email address or password is wrong, an alert message will remind the user to check the typing (Figure 6).

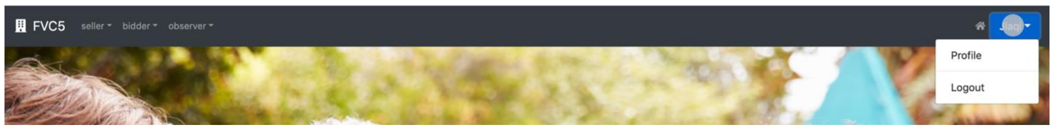
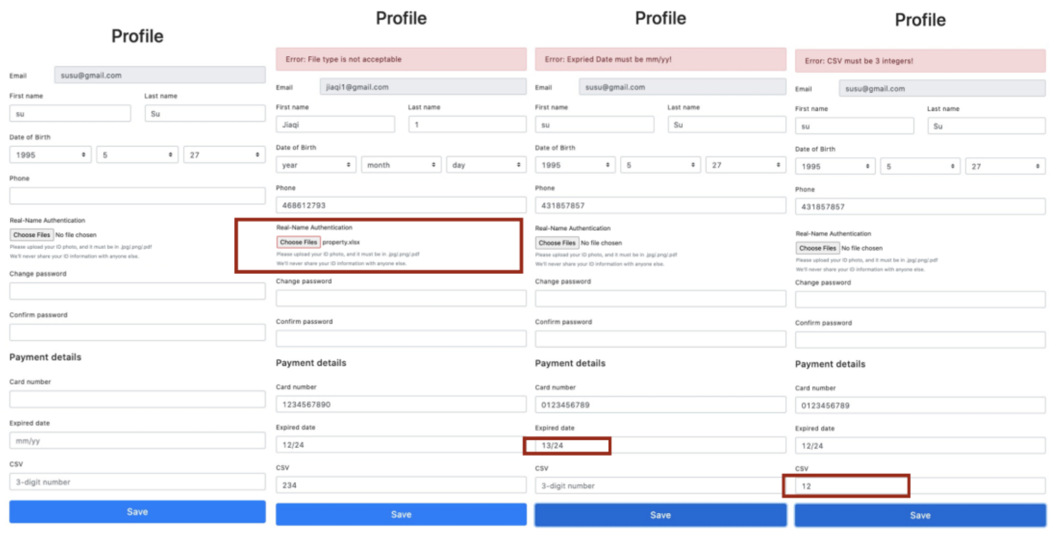


Figure 7. The after-login version Navbar

After successful login, the Navbar will be changed to version 2 (Figure 7). On the left side of the Navbar, there are three drop-down menus indicating user types, each of which links to the corresponding users’ dashboards (will be described in section 2.3). On the right side, it will show the username, and the users can click the button displaying the username to choose to log out or update the individual profile.

2.2.3 Update Profile


Profile

Email: susu@gmail.com

First name: su | Last name: Su

Date of Birth: 1995 | 5 | 27

Phone:

Real-Name Authentication: Choose Files No file chosen

Please upload your ID photo, and it must be in .jpg/.png/.pdf

We'll never share your ID information with anyone else.

Change password:

Confirm password:

Payment details

Card number:

Expired date: mm/yy

CSV: 3-digit number

Save

Profile

Error: File type is not acceptable

Email: jiaqi1@gmail.com

First name: Jiaqi | Last name: 1

Date of Birth: year | month | day

Phone: 468612793

Real-Name Authentication: Choose Files property.xlsx

Please upload your ID photo, and it must be in .jpg/.png/.pdf

We'll never share your ID information with anyone else.

Change password:

Confirm password:

Payment details

Card number: 1234567890

Expired date: 12/24

CSV: 234

Save

Profile

Error: Expried Date must be mm/yy!

Email: susu@gmail.com

First name: su | Last name: Su

Date of Birth: 1995 | 5 | 27

Phone: 431857857

Real-Name Authentication: Choose Files No file chosen

Please upload your ID photo, and it must be in .jpg/.png/.pdf

We'll never share your ID information with anyone else.

Change password:

Confirm password:

Payment details

Card number: 0123456789

Expired date: 13/24

CSV: 3-digit number

Save

Profile

Error: CSV must be 3 integers!

Email: susu@gmail.com

First name: su | Last name: Su

Date of Birth: 1995 | 5 | 27

Phone: 431857857

Real-Name Authentication: Choose Files No file chosen

Please upload your ID photo, and it must be in .jpg/.png/.pdf

We'll never share your ID information with anyone else.

Change password:

Confirm password:

Payment details

Card number: 0123456789

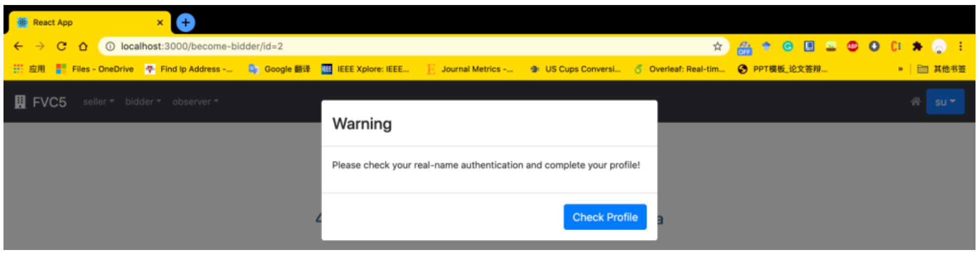
Expired date: 12/24

CSV: 12

Save

Figure 8. The profile page with the error alert messages

The '*profile*' page collects the user individual information. The users are able to update their profile and see their up-to-date profile in this page. At the first time the user opens this page, the information which filled in the '*register*' page will return to this page from the back-end (Figure 8).



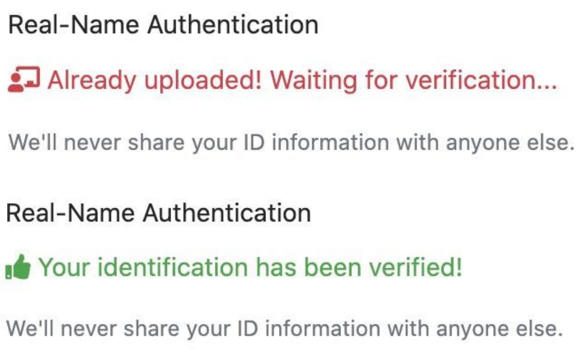
Warning

Please check your real-name authentication and complete your profile!

Check Profile

Figure 9. The uncompleted profile and real-name authentication warning

The users have to upload ID photo and payment details if they want to post or bid an auction (Figure 9). The upload function only accepts .jpg, .png or .pdf file, the expired date field will check the date and respond with an alert message if invalid, CSV field needs 3 digital numbers (Figure 8). If the above conditions are not satisfied, a corresponding alert message will appear.


Real-Name Authentication

Already uploaded! Waiting for verification...

We'll never share your ID information with anyone else.

Real-Name Authentication

Your identification has been verified!

We'll never share your ID information with anyone else.

Figure 10. Different messages after uploading Authentication in the profile page

After uploading the personal ID information, the user is not allowed to upload it again afterwards. The manager of this website will check the user identity manually, and a message will inform the user whether the authentication is verified or not, which is shown in Figure 10. If the user has not passed the authentication, he does not have the authorization to post or bid an auction.

2.3 User Dashboard

2.3.1 Seller

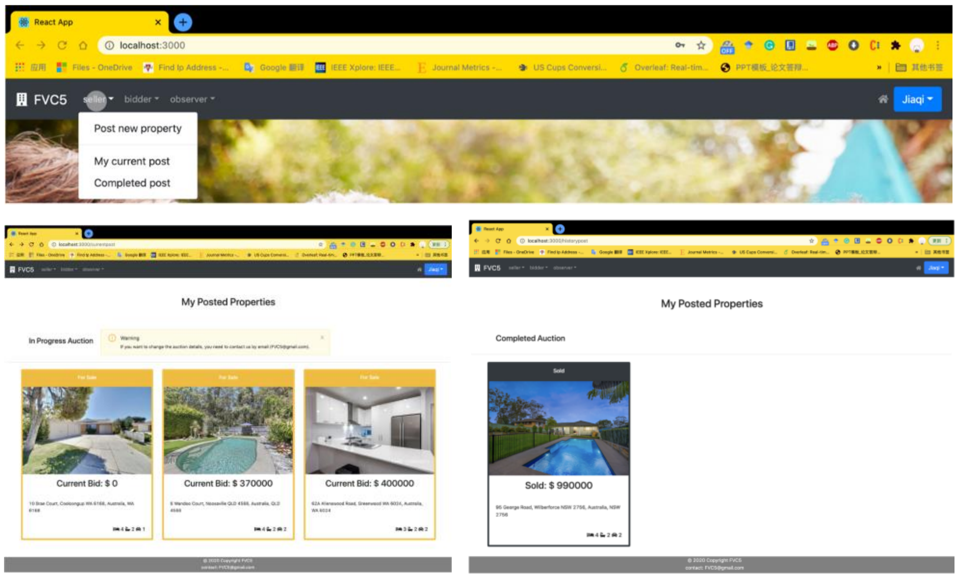


Figure 11. The dashboard for sellers

The '*seller*' drop-down menu on the Navbar allows users to post a new property or to browse their posted auctions. '*My current post*' button links to the '*currentpost*' page, which shows the auctions (posted by the current user) in progress. The ‘*completed post*’ button links to the ‘*historypost*’ page, which shows the auctions (posted by the current user) that have completed (Figure 11).

Once the auction has posted, the user cannot modify or revoke it. At the top of the ‘*currentpost*’ page, there is a message indicating sellers to contact the website manage by email if they want to modify or revoke the auction.

2.3.2 Bidder

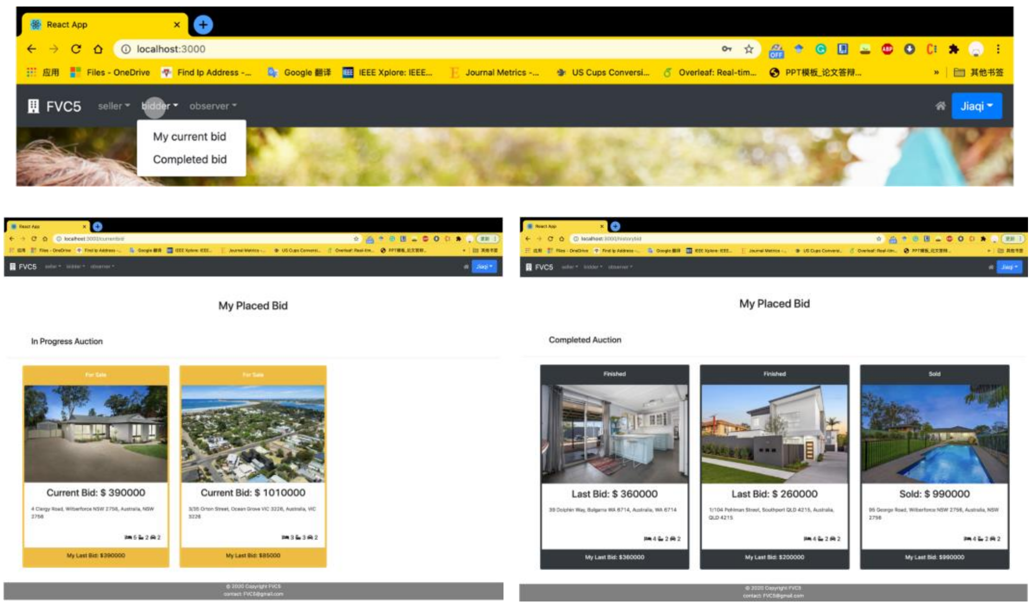
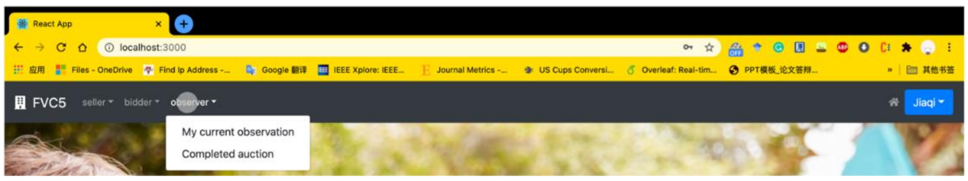


Figure 12. The dashboard for bidders

The ‘*bidder*’ drop-down menu on the Navbar has two buttons: the ‘*My current bid*’ button links to the ‘*currentbid*’ page, the ‘*Completed bid*’ button links to the ‘*historybid*’ page, and these two pages display the corresponding auctions (bade by the current user) in different states respectively (Figure 12). On these pages, the auction cards have different formats compared to other pages. The bidder can check the bid price that he previously placed.

2.3.3 Observer



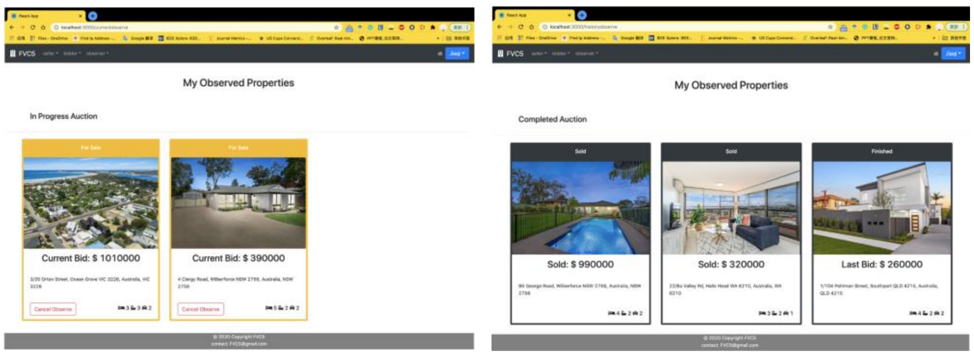
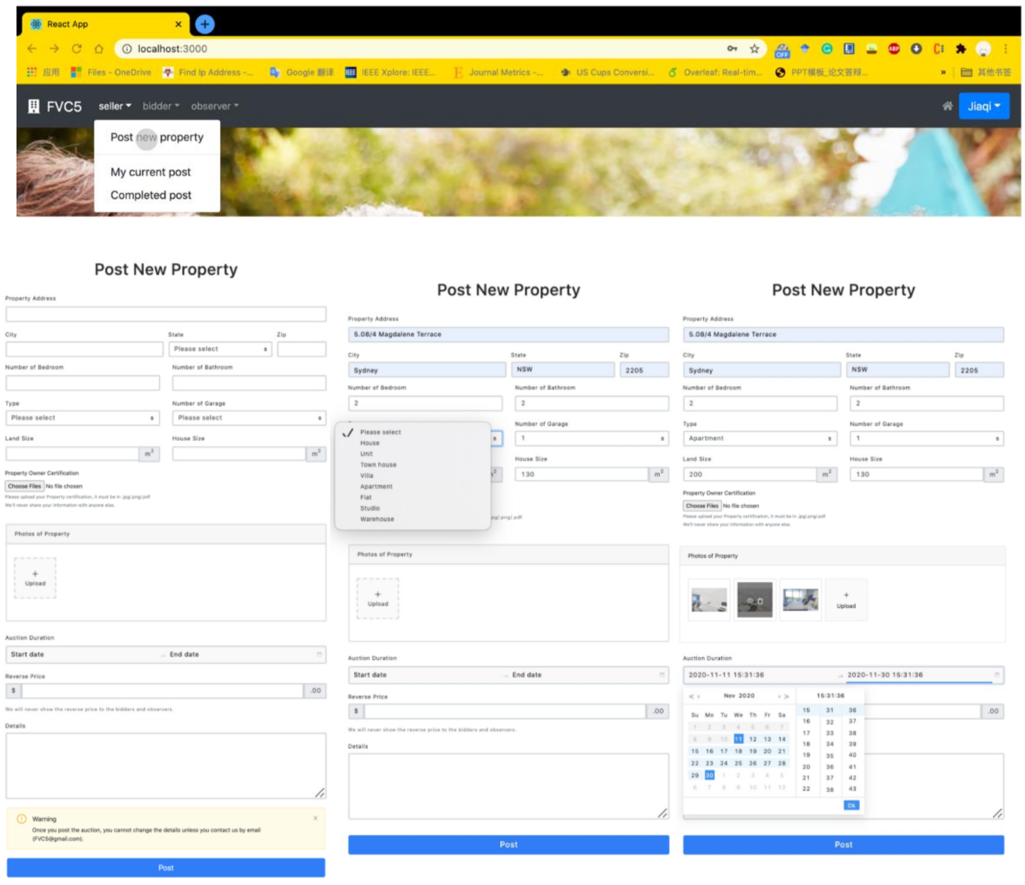


Figure 13. The dashboard for observers

The ‘*observer*’ drop-down menu also shows two options, ‘*My current observation*’ and ‘*Completed auction*’, which will show the in-progress auctions and completed auctions (observed by the current user) in the corresponding page, respectively (Figure 13). In the current observation page, the user can cancel the observation of the observed auctions on the card.

2.4 Post Property



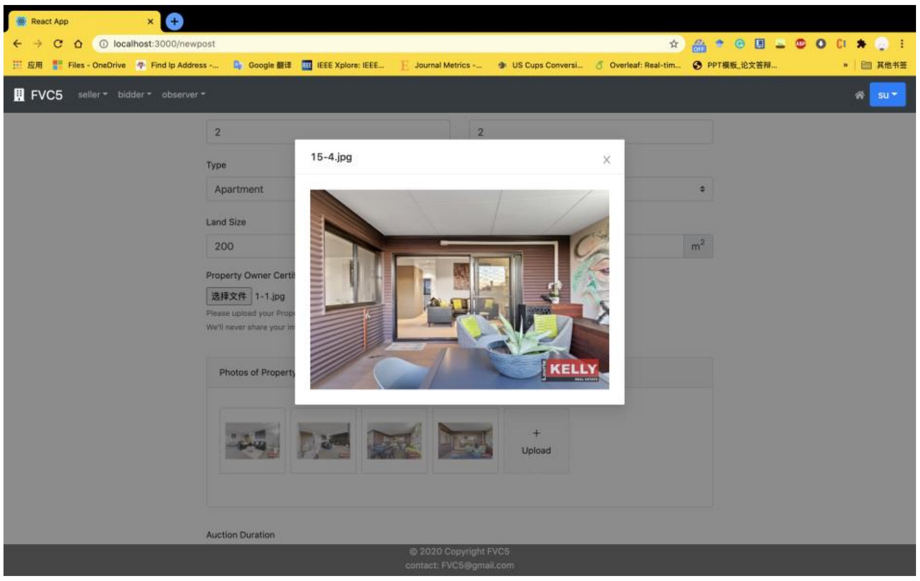
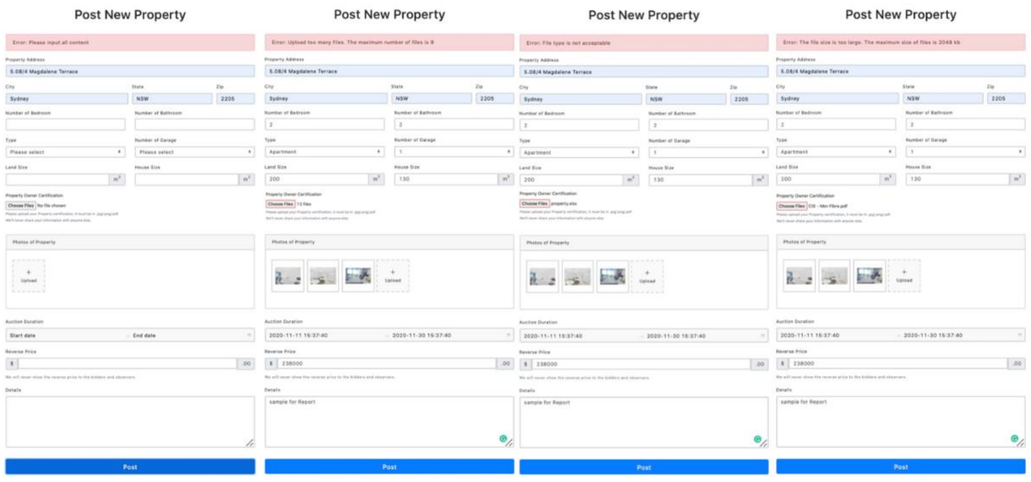


Figure 14. The picture preview in the post property page

If a user wants to post an auction on this website, he can click the seller drop-down menu on the navbar and click the '*post new property*' button to jump to the '*newpost*' page shown in Figure 14. On this page, the seller can post the property information (address, room number, property type, property size, pictures) as well as the auction details (reserve price and starting/end time). Besides, the seller has to upload the Owner Certification for verifying the authenticity of the property.

By clicking the eye icon on the property thumbnail, a high-definition image will pop up. The reserve price, which is only used for comparing with the last bid to determine whether the auction is sold successfully or not, is hidden from the bidders and observers.


Post New Property

Error: Please input all content

Post New Property

Error: Upload too many files. The maximum number of files is 8

Post New Property

Error: File type is not acceptable

Post New Property

Error: The file size is too large. The maximum size of files is 2048 kb

Figure 15. The error alert messages in the post property page

There is an alert message to tell the sellers to be careful of the posting information at the bottom of '*newpost*' page, since the posted auction cannot be modified and revoked by the seller's side. If the information needs to be modified, the seller has to contact the website manager by sending email.

Some situations listed below will trigger the error alert message (Figure 15):

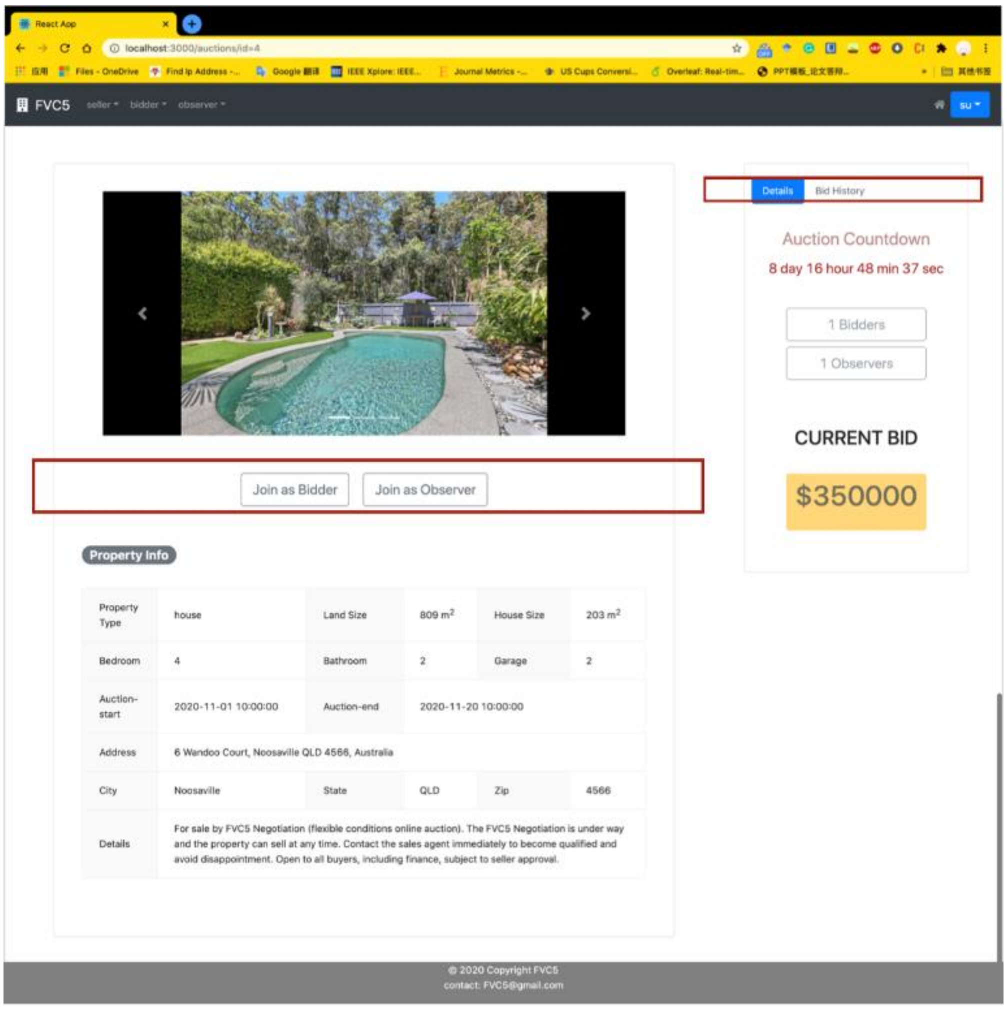
a. Some fields in the form are empty

b. The number of Property Owner Certification files exceeds the upper limit (9 files)

c. The uploaded file type is unaccepted (.jpg, .png, .pdf)

d. The file size is larger than 2048 kB

2.5 Auction details page



Property Type	house	Land Size	809 m^2	House Size	203 m^2
Bedroom	4	Bathroom	2	Garage	2
Auction-start	2020-11-01 10:00:00	Auction-end	2020-11-20 10:00:00		
Address	6 Wandoo Court, Noosaville QLD 4566, Australia				
City	Noosaville	State	QLD	Zip	4566
Details	For sale by FVC5 Negotiation (flexible conditions online auction). The FVC5 Negotiation is under way and the property can sell at any time. Contact the sales agent immediately to become qualified and avoid disappointment. Open to all buyers, including finance, subject to seller approval.				

Figure 16. Auction details page

Every auction has a page describing its property details as well as the auction date (Figure 16). When opening an auction details page for a specific auction, the back-end database will return the corresponding property information to the front-end, and then the front-end will render the information to this page. The sliding window will cycle through photos automatically, and the user can also click the left and right arrow to quickly view the pictures.

If the user is the seller of the current property, the '*Join as Bidder*' and '*Join as Observer*' buttons will be not shown in this page.

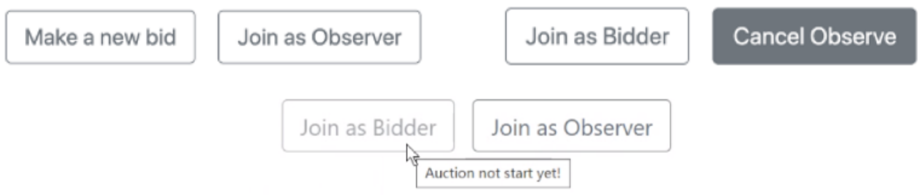


Figure 17. The difference of two buttons on the auction details page

If the user has registered a bidder for the current auction, the left button will change to '*Make a new bid*' instead of the '*Join as Bidder*' (Figure 17), and the bidder can make a higher bid by clicking this button. If the user has observed the current property, the right button will change to '*Cancel Observe*' (Figure 17), the observer can unfollow the property by clicking this button if he is not interested in this auction.

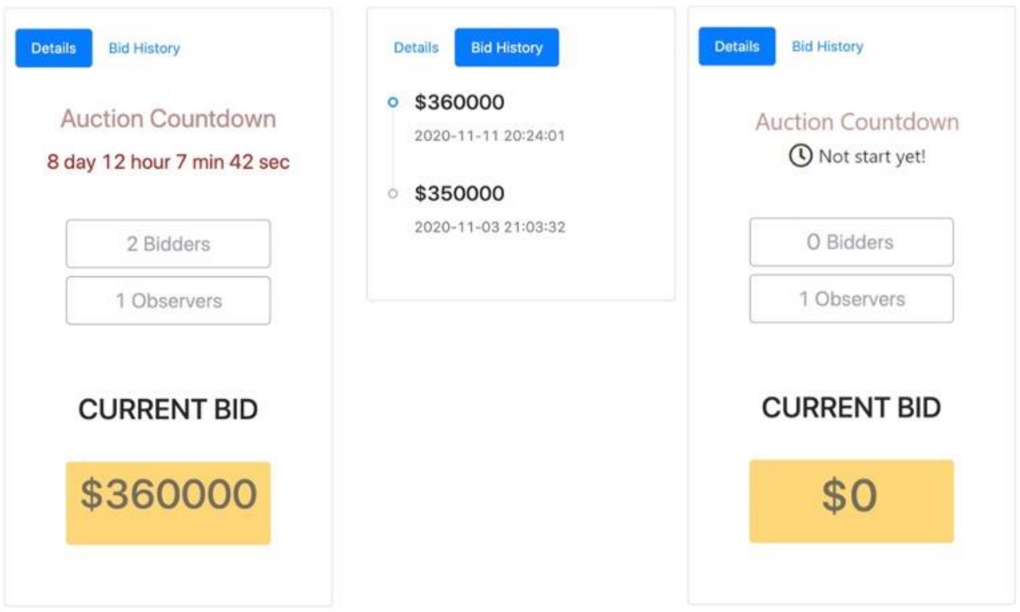
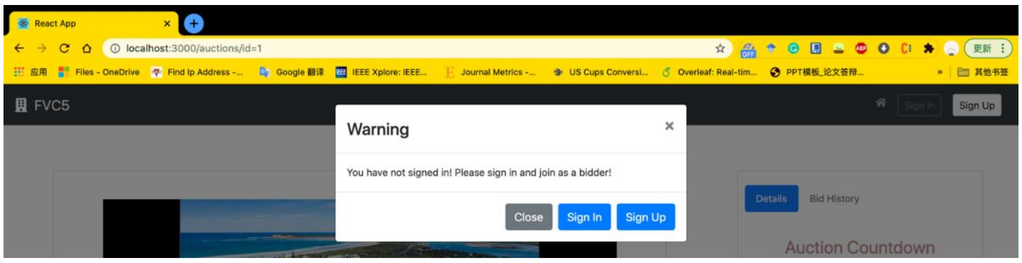


Figure 18. Auction details and bid history on the auction details page

On the top right of the auction details page, there is an area displaying the '*Details*' and '*Bid History*' of this auction (Figure 18). The '*Details*' shows the count down, number of bidders, number of observers and current bid. If the auction has not started, the auction details section will show '*Not start yet!*'. Users cannot join as a bidder but still can join as observer if auction has not started. The '*Bid History*' records the bids and the time when the bid was placed. The bid history is only shown to the users who are involved in this auction, including the seller, bidders and observers, and is hidden to other users.



Warning

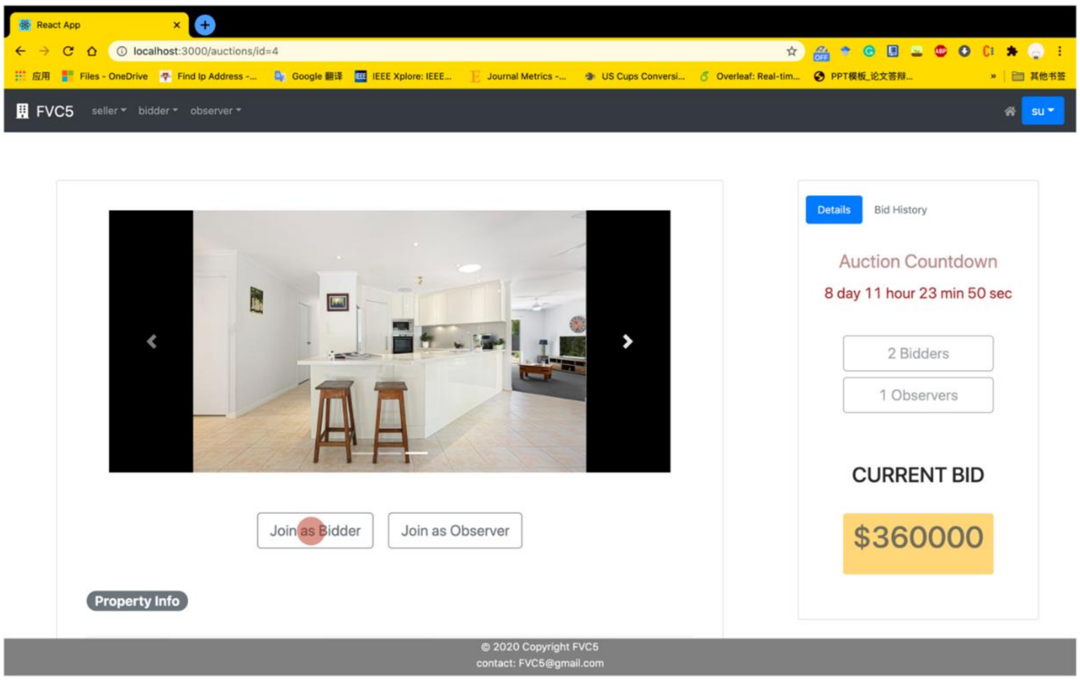
You have not signed in! Please sign in and join as a bidder!

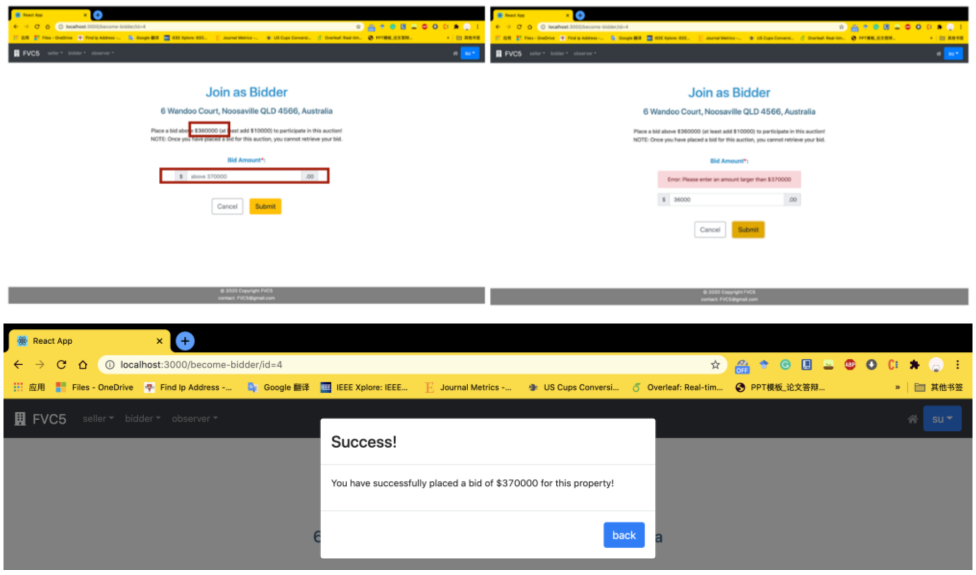
Close Sign In Sign Up

Figure 19. Warning message for unlogged-in users

The website will throw a warning message if an unlogged-in user clicks the '*Join as Bidder*' or '*Join as Observer*' buttons, and the page will force the visitor to sign in or sign up (Figure 19).

2.6 Bid an Auction




Join as Bidder

6 Wandoo Court, Noosaville QLD 4566, Australia

Place a bid above $360000 (at least add $10000) to participate in this auction!
NOTE: Once you have placed a bid for this auction, you cannot retrieve your bid.

Bid Amount*:

$ above 370000 .00

Error: Please enter an amount larger than $370000

$ 36000 .00

Success!

You have successfully placed a bid of $370000 for this property!

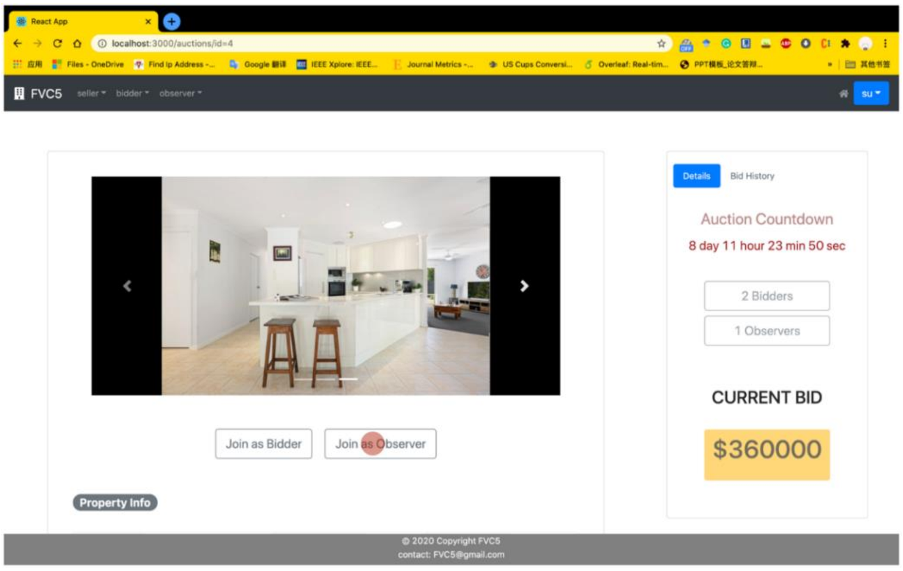
© 2020 Copyright FVC5
contact: FVC5@gmail.com

Figure 20. The process of making a bid

By clicking the '*Join as Bidder*' button, the authorized user can choose to place a new bid for this auction. As the Figure 20 shown, in this page, the user can check the property's address and the current bid of the auction, and he can place a bid greater than or equal to the current bid + $10000.

If the user has not completed the profile or the ID authentication has not been verified, he cannot join as a bidder (with an alert window). If the amount inputted by an authorized user does not meet the requirement, an error message will appear. Otherwise, a new bid will be added to this auction.

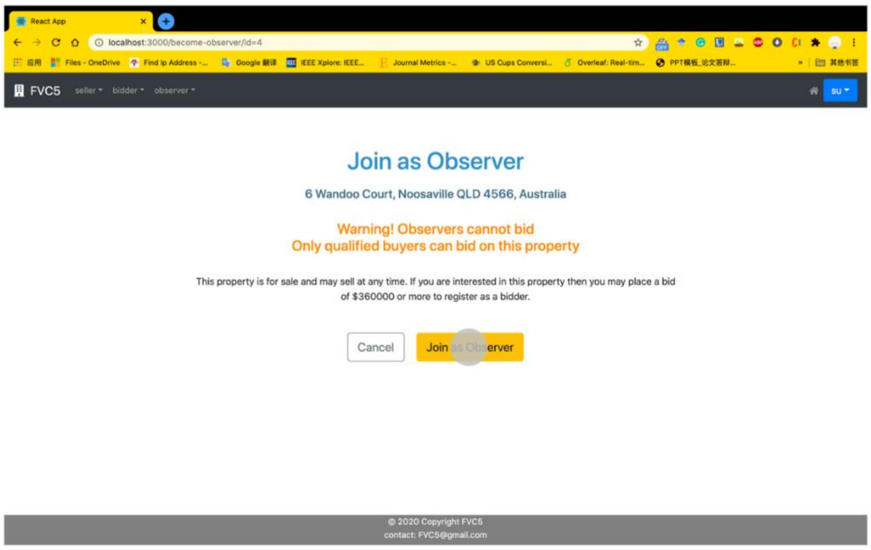
2.7 Observe a Property


Auction Countdown
8 day 11 hour 23 min 50 sec

2 Bidders
1 Observers

CURRENT BID
$360000

© 2020 Copyright FVC5
contact: FVC5@gmail.com



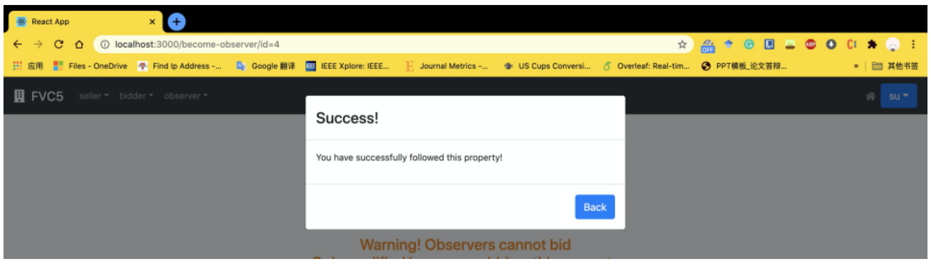
Join as Observer

6 Wandoo Court, Noosaville QLD 4566, Australia

Warning! Observers cannot bid
Only qualified buyers can bid on this property

This property is for sale and may sell at any time. If you are interested in this property then you may place a bid of $360000 or more to register as a bidder.

© 2020 Copyright FVC5
contact: FVC5@gmail.com



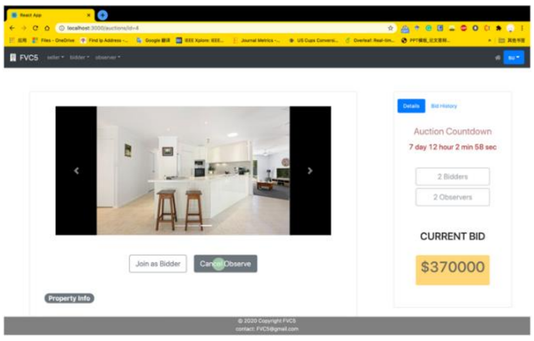
Success!

You have successfully followed this property!

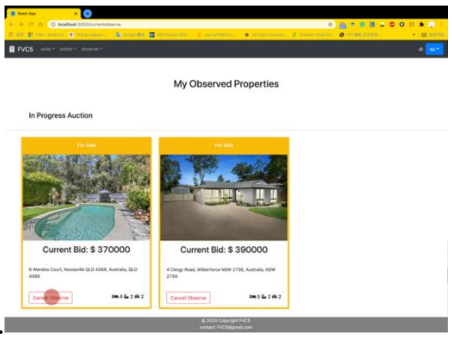
Warning! Observers cannot bid

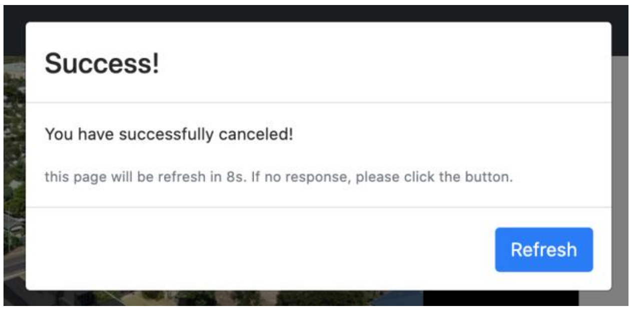
Figure 21. The process of joining as an observer

By clicking the '*Join as Observer*', the user will go to the '*become-observer/id=?*' page. This page displays the property address and reminds the users of the current bid. After becoming an observer of this property, the users can see the bid history on the auction details page.



…..




Success!

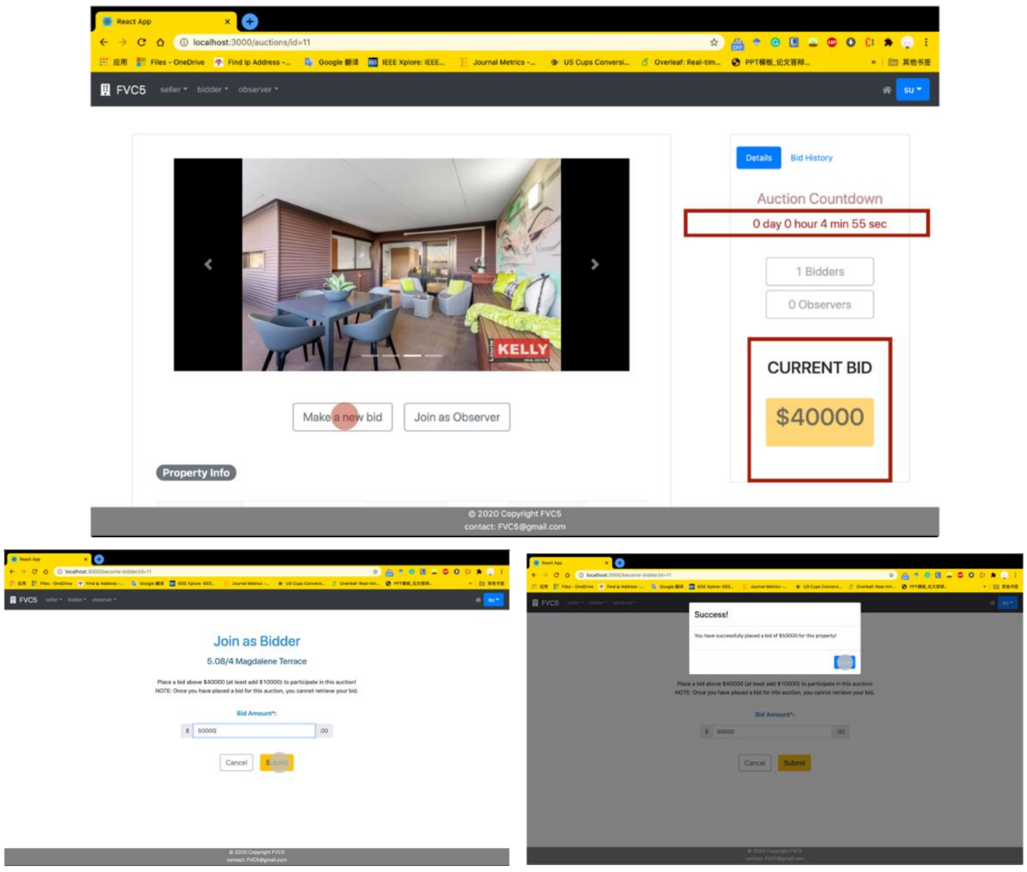
You have successfully canceled!

this page will be refresh in 8s. If no response, please click the button.

Figure 22. The process of cancelling observation on the auction details page

If a user does not want to observe the auction anymore, our website allows users to cancel the observation. There are two ways to cancel the observation. One is to click the '*Cancel Observe*' button on the auction details page. After cancelling observation successfully, the system will announce the cancel result in a pop-up window, the current page will automatically refresh in 8s if the user does not close the pop-up window (Figure 22). Another one is to click the '*Cancel Observe*' button on the cards in the '*my current observation*' which is mentioned before.

2.8 Auction End Time Extension



Place a bid above $40000 (at least add $10000) to participate in this auction!
NOTE: Once you have placed a bid for this auction, you cannot retrieve your bid.

You have successfully placed a bid of $50000 for this property!

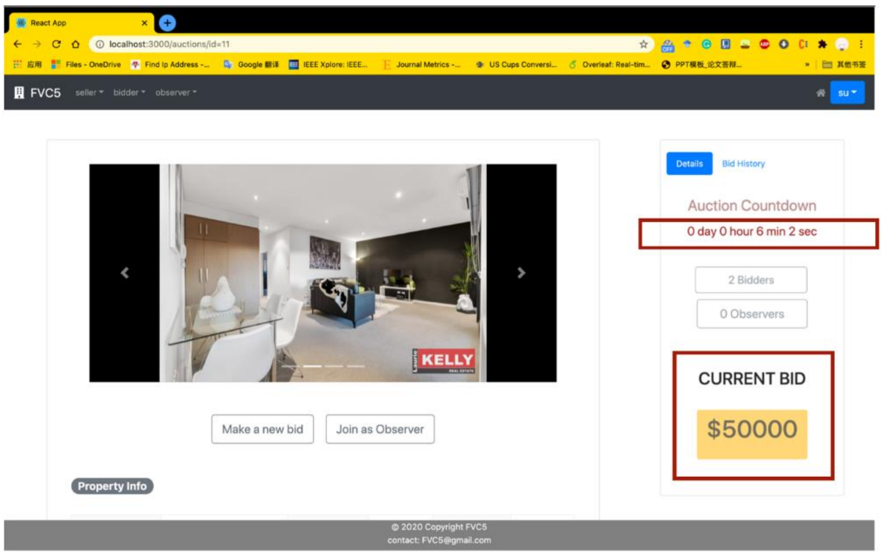
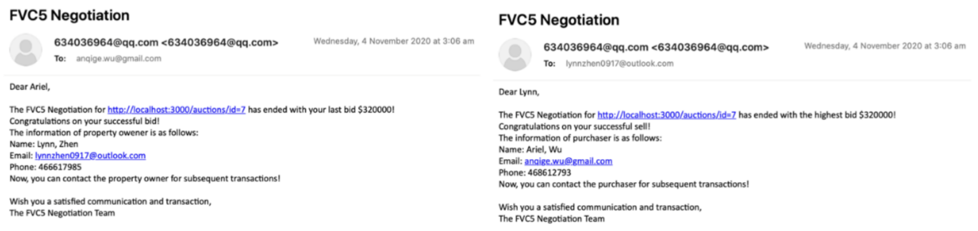


Figure 23. Adding 2 more minutes if a user bids in last five minutes

If a bidder makes a new bid during the last 5 minutes of the auction, the auction end time will extend 2 minutes automatically, which allows other bidders to place bids for this auction (Figure 23).

2.9 Completed Auction and Notifications



FVC5 Negotiation

634036964@qq.com <634036964@qq.com> Wednesday, 4 November 2020 at 3:06 am
To: anqige.wu@gmail.com

Dear Ariel,

The FVC5 Negotiation for http://localhost:3000/auctions/id=7 has ended with your last bid $320000!
Congratulations on your successful bid!
The information of property owener is as follows:
Name: Lynn, Zhen
Email: lynnzhen0917@outlook.com
Phone: 466617985
Now, you can contact the property owner for subsequent transactions!

Wish you a satisfied communication and transaction,
The FVC5 Negotiation Team

FVC5 Negotiation

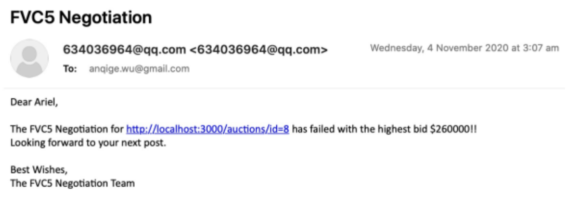
634036964@qq.com <634036964@qq.com> Wednesday, 4 November 2020 at 3:06 am
To: lynnzhen0917@outlook.com

Dear Lynn,

The FVC5 Negotiation for http://localhost:3000/auctions/id=7 has ended with the highest bid $320000!
Congratulations on your successful sell!
The information of purchaser is as follows:
Name: Ariel, Wu
Email: anqige.wu@gmail.com
Phone: 468612793
Now, you can contact the purchaser for subsequent transactions!

Wish you a satisfied communication and transaction,
The FVC5 Negotiation Team

Figure 24. The emails sent to seller and last bidder if the auction success



FVC5 Negotiation

634036964@qq.com <634036964@qq.com> Wednesday, 4 November 2020 at 3:07 am
To: anqige.wu@gmail.com

Dear Ariel,

The FVC5 Negotiation for http://localhost:3000/auctions/id=8 has failed with the highest bid $260000!!
Looking forward to your next post.

Best Wishes,
The FVC5 Negotiation Team

Figure 25. The email sent to seller if the auction failed

When the auction has completed, there are two states for this auction, sold and finished.

If the last bid is higher or equal to the seller reserved price, the auction is successfully sold. The system will send results to the seller and the winning bidder by email, which helps them exchange the contact information to continue the subsequent transaction (Figure 24).

If the last bid is lower than the seller reserved price, the auction is finished (not sold). The system will only notify the highest price to the seller (Figure 25), and no bidders will get this property.

2.10 Recommendation

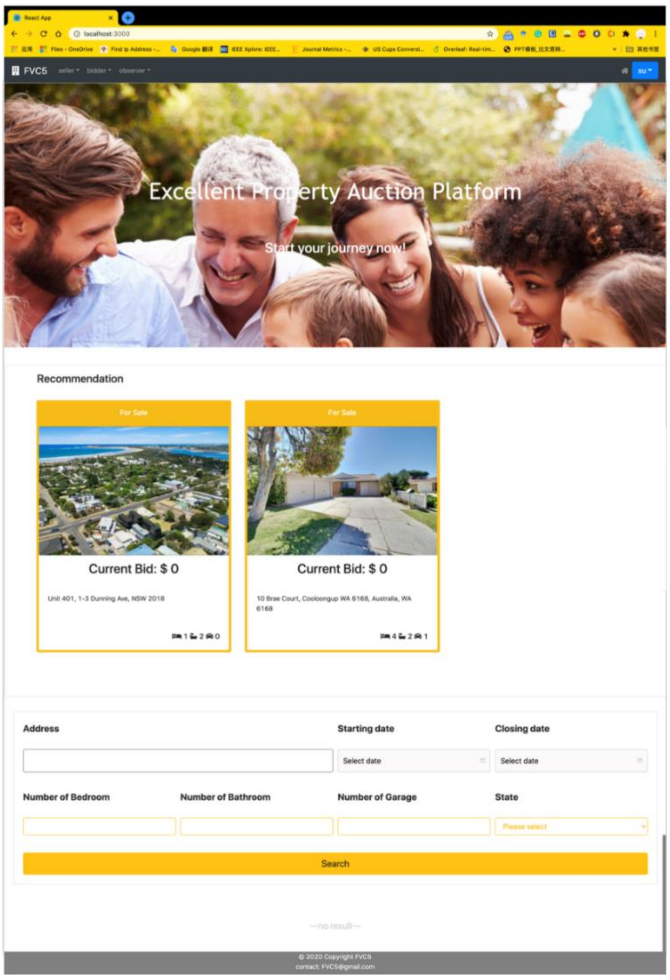
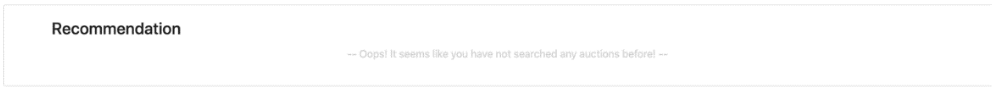


Figure 26. The logged-in homepage with recommendation box

The recommendation function is based on the user's search history. The back-end database records the behavior of the user and returns the recommendations that the user may be interested in on the homepage.

The system provides accurate recommendation results for the users, which takes 5 filters into consideration (address, number of bedrooms, number of bathrooms, number of garage and state). The user can get the recently posted in-progress auctions which are not focused before, according to their previous search history (Figure 26). It should be noticed that the recommended auctions will not include the auctions that the users have involved in. If the number of recommended auctions is greater than 3, the system will choose 3 auctions randomly to display on the homepage. If the user only filtered the auction dates before, the system would select the most recently posted auctions for the user, since the auction date might be useless if the date is expired.

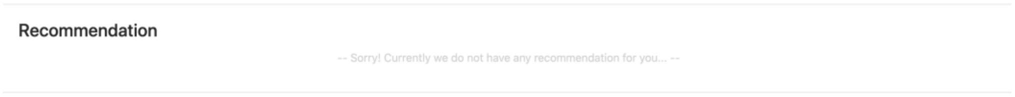


Recommendation

-- Oops! It seems like you have not searched any auctions before! --

Figure 27. Empty recommendation box

If the user does not search any auctions by filters, the recommendation section will notify the user with the message shown in Figure 27.



Recommendation

-- Sorry! Currently we do not have any recommendation for you... --

Figure 28. No recommendation

If the user has search history while the recommendation algorithm does not have any matching results for the user, the recommendation section will show a message in Figure 28.

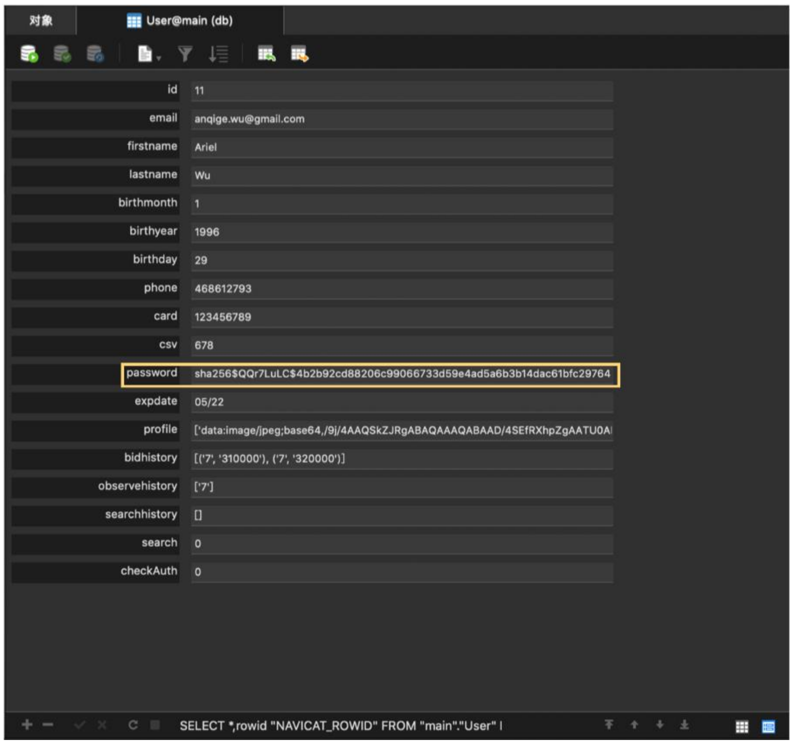
2.11 Logout

The user can log out by clicking the '*Logout*' button which is in the username's drop-down menu (Figure 7).

3. Implementation Challenges

3.1 Front-end and Back-end

3.1.1 Encoding password and images

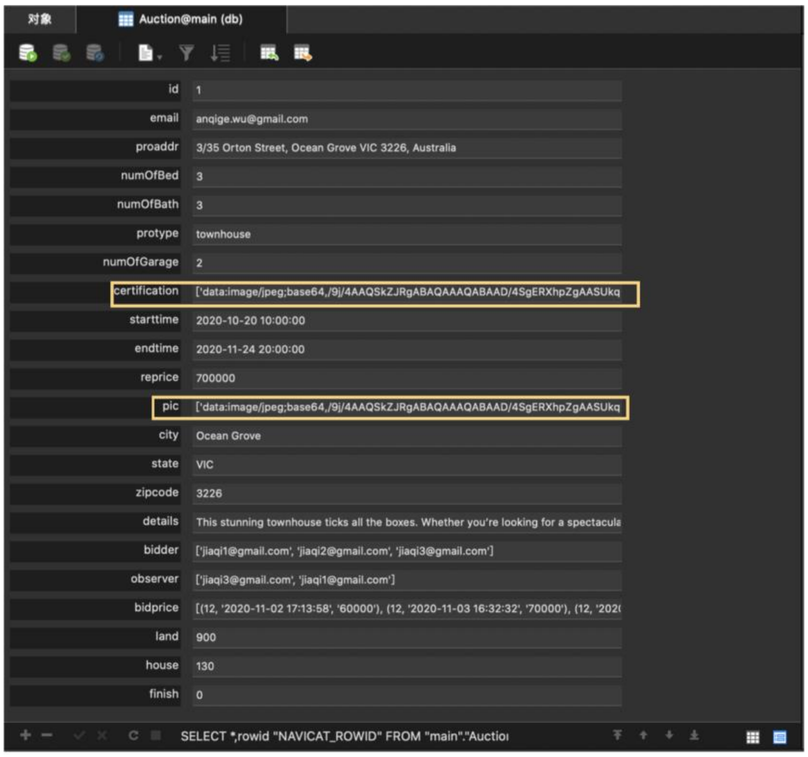

对象 | User@main (db)

id	11
email	anqige.wu@gmail.com
firstname	Ariel
lastname	Wu
birthmonth	1
birthyear	1996
birthday	29
phone	468612793
card	123456789
csv	678
password	sha256$QQr7LuLC$4b2b92cd88206c99066733d59e4ad5a6b3b14dac61bfc29764
expdate	05/22
profile	['data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/4SEfRXhpZgAATU0A
bidhistory	[('7', '310000'), ('7', '320000')]
observehistory	['7']
searchhistory	[]
search	0
checkAuth	0

```
SELECT *,rowid "NAVICAT_ROWID" FROM "main"."User" l
```

Figure 29. The user table of database.

We use SHA256 Hex digit to encode the user password to protect the user information safety. The confirm password function decode the hash coded password to compare with the new typed '*confirm password*'.


对象 | Auction@main (db)

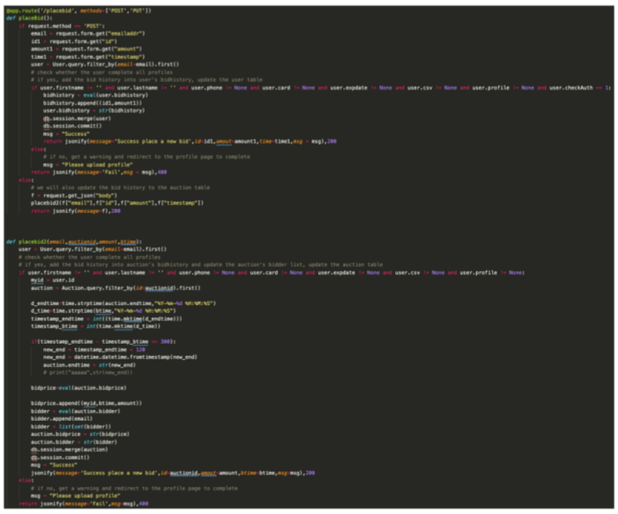
id	1
email	anqige.wu@gmail.com
proaddr	3/35 Orton Street, Ocean Grove VIC 3226, Australia
numOfBed	3
numOfBath	3
protype	townhouse
numOfGarage	2
certification	['data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/4SgERXhpZgAASUkq
starttime	2020-10-20 10:00:00
endtime	2020-11-24 20:00:00
reprice	700000
pic	['data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/4SgERXhpZgAASUkq
city	Ocean Grove
state	VIC
zipcode	3226
details	This stunning townhouse ticks all the boxes. Whether you're looking for a spectacula
bidder	['jiaqi1@gmail.com', 'jiaqi2@gmail.com', 'jiaqi3@gmail.com']
observer	['jiaqi3@gmail.com', 'jiaqi1@gmail.com']
bidprice	[(12, '2020-11-02 17:13:58', '60000'), (12, '2020-11-03 16:32:32', '70000'), (12, '202(
land	900
house	130
finish	0

```
SELECT *,rowid "NAVICAT_ROWID" FROM "main"."Auctio
```

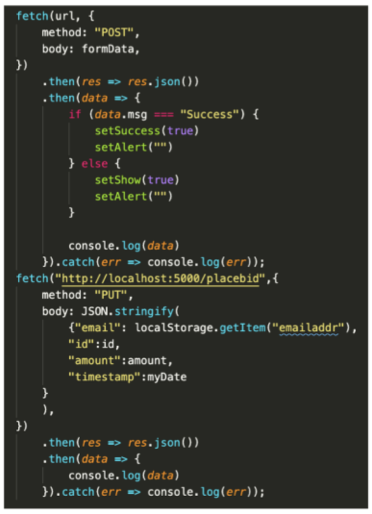
Figure 30. The auction table of database

We import FileBase64 model from ‘react-file-base64’ library, and use this model covert the files to 64 based code for store in the back-end database. If the front-end needs to extract the file, it will extract the 64-base code from the database, and convert the code back to file through a third-party application.

3.1.2 SQLite cannot update two tables synchronously



```python
@app.route('/placebid', methods=['POST','PUT'])
def placeBid():
    if request.method == 'POST':
        email = request.form.get("emailaddr")
        id1 = request.form.get("id")
        amount1 = request.form.get("amount")
        time1 = request.form.get("timestamp")
        user = User.query.filter_by(email=email).first()
        # check whether the user complete all profiles
        # if yes, add the bid history into user's bidhistory, update the user table
        if user.firstname != '' and user.lastname != '' and user.phone != None and user.card != None and user.expdate != None and user.csv != None and user.profile != None and user.checkAuth == 1:
            bidhistory = eval(user.bidhistory)
            bidhistory.append((id1,amount1))
            user.bidhistory = str(bidhistory)
            db.session.merge(user)
            db.session.commit()
            msg = "Success"
            return jsonify(message='Success place a new bid',id=id1,amout=amount1,time=time1,msg = msg),200
        else:
            # if no, get a warning and redirect to the profile page to complete
            msg = "Please upload profile"
        return jsonify(message='Fail',msg = msg),400
    else:
        # we will also update the bid history to the auction table
        f = request.get_json("body")
        placebid2(f["email"],f["id"],f["amount"],f["timestamp"])
        return jsonify(message=f),200


def placebid2(email,auctionid,amount,btime):
    user = User.query.filter_by(email=email).first()
    # check whether the user complete all profiles
    # if yes, add the bid history into auction's bidhistory and update the auction's bidder list, update the auction table
    if user.firstname != '' and user.lastname != '' and user.phone != None and user.card != None and user.expdate != None and user.csv != None and user.profile != None:
        myid = user.id
        auction = Auction.query.filter_by(id=auctionid).first()

        d_endtime=time.strptime(auction.endtime,"%Y-%m-%d %H:%M:%S")
        d_time=time.strptime(btime,"%Y-%m-%d %H:%M:%S")
        timestamp_endtime = int((time.mktime(d_endtime)))
        timestamp_btime = int(time.mktime(d_time))

        if(timestamp_endtime - timestamp_btime <= 300):
            new_end = timestamp_endtime + 120
            new_end = datetime.datetime.fromtimestamp(new_end)
            auction.endtime = str(new_end)
            # print("aaaaa",str(new_end))

        bidprice=eval(auction.bidprice)

        bidprice.append((myid,btime,amount))
        bidder = eval(auction.bidder)
        bidder.append(email)
        bidder = list(set(bidder))
        auction.bidprice = str(bidprice)
        auction.bidder = str(bidder)
        db.session.merge(auction)
        db.session.commit()
        msg = "Success"
        jsonify(message='Success place a new bid',id=auctionid,amout=amount,btime=btime,msg=msg),200
    else:
        # if no, get a warning and redirect to the profile page to complete
        msg = "Please upload profile"
    return jsonify(message='Fail',msg=msg),400
```



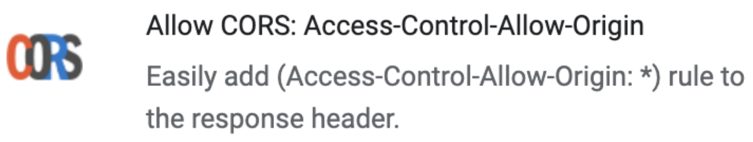
```javascript
fetch(url, {
    method: "POST",
    body: formData,
})
    .then(res => res.json())
    .then(data => {
        if (data.msg === "Success") {
            setSuccess(true)
            setAlert("")
        } else {
            setShow(true)
            setAlert("")
        }

        console.log(data)
    }).catch(err => console.log(err));
fetch("http://localhost:5000/placebid",{
    method: "PUT",
    body: JSON.stringify(
        {"email": localStorage.getItem("emailaddr"),
        "id":id,
        "amount":amount,
        "timestamp":myDate
    }
    ),
})
    .then(res => res.json())
    .then(data => {
        console.log(data)
    }).catch(err => console.log(err));
```

Figure 31. Part of auth.py and afterinterest.js files corresponding to update database

When users make a bid, the user table in the database needs to record the property id and bid number in the ‘*bidhistory*’ attribute (Figure 29), and the auction table needs to record the bid price and bidder email in the ‘*bidprice*’ and ‘*bidder*’ attributes (Figure 30). Due to the SQLite will trigger a file lock when it modifies the database, other thread cannot make a change on the database at the same time, the two tables cannot update by one operation. To solve this problem, the developers set two functions to call two flask requests.

3.1.3 The front and back end cannot be connected

We insert the Cross-origin resource sharing plugin. It is a mechanism that allows requesting restricted resources on a web page from another domain other than the domain where the first resource is provided. Web pages can freely embed cross-domain images, style sheets, scripts, iframes and videos.


Allow CORS: Access-Control-Allow-Origin

Easily add (Access-Control-Allow-Origin: *) rule to the response header.

Figure 32. The Cross-origin resource sharing plugin

CORS can help us connect the front and back ends, and the system can run smoothly.

3.1.4 Recommendation algorithm

According to the user's search history, recommend the interested properties on the main interface (showed in Figure 26). All the recommended real estate information has not been viewed by users and is not published by themselves.

We classify users into three categories to recommend properties. The first category is users who have not searched for any real estate information on the web, the second category is users who have only searched for real estate start and end time attributes, and the third category is users who have searched for attributes other than time attributes.

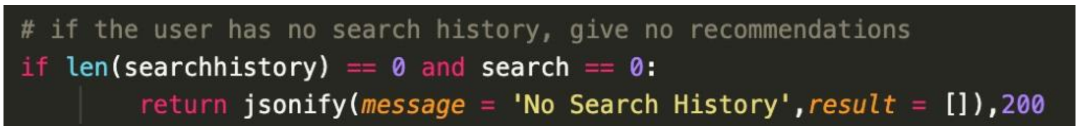

```
# if the user has no search history, give no recommendations
if len(searchhistory) == 0 and search == 0:
        return jsonify(message = 'No Search History',result = []),200
```

Figure 33. Code for empty recommendation

For the first type of users, the system will return a message to tell the front-end that the user does not have any research history and the system will not be able to recommend real estate for the user. After the front-end receives the back-end information, it will display the information in Figure 27 on the web page.

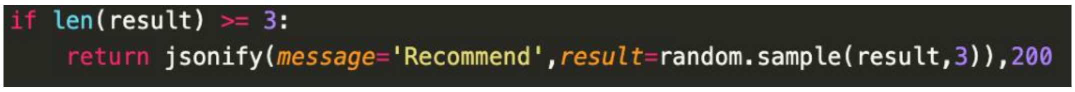
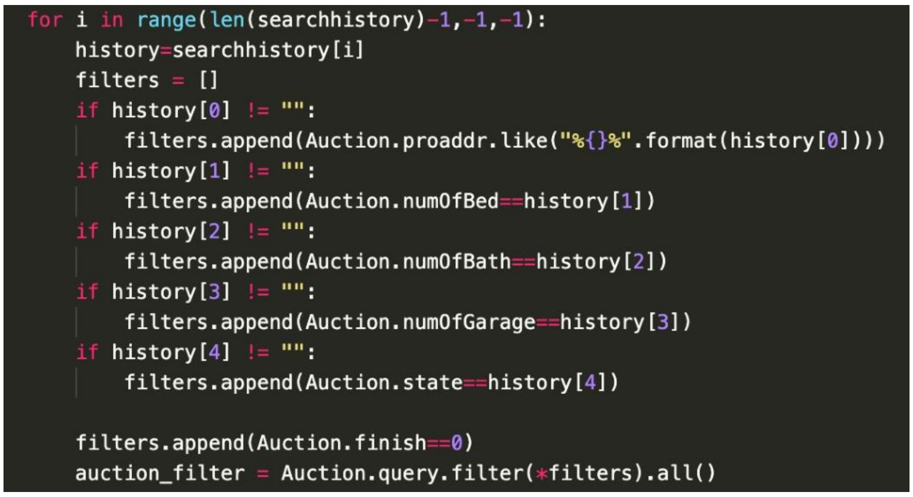

```
if len(result) >= 3:
    return jsonify(message='Recommend',result=random.sample(result,3)),200
```

Figure 34. Recommend according to the time attribute

For the second type of user, the system will recommend the latest 3 properties that meet his search time requirements. We put the data filtered by the system in the result list and return to the front-end.

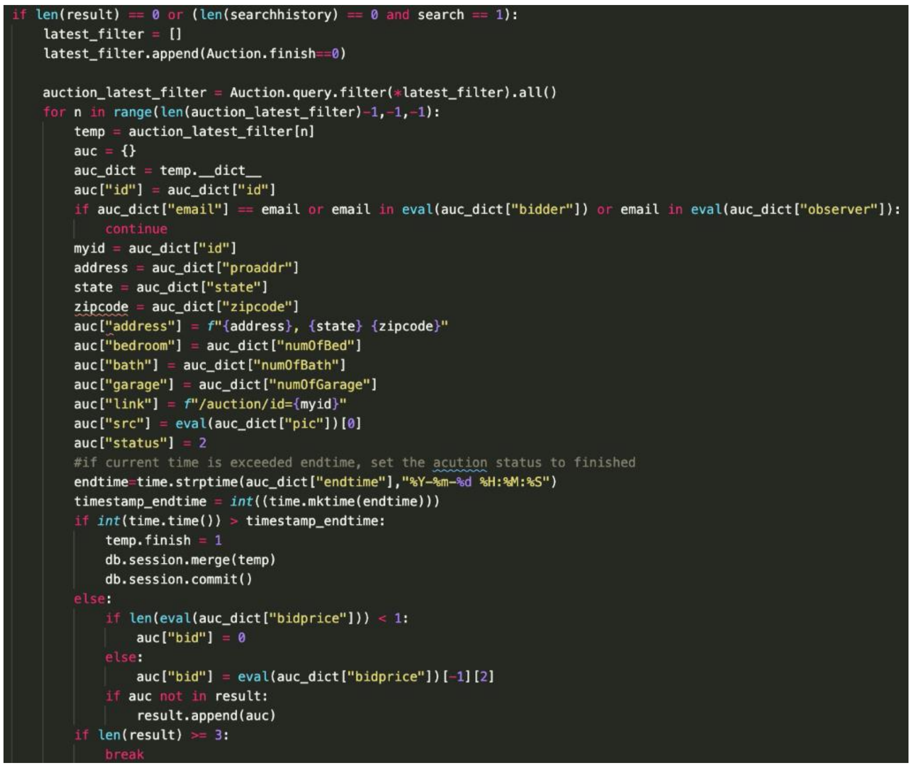


```python
for i in range(len(searchhistory)-1,-1,-1):
    history=searchhistory[i]
    filters = []
    if history[0] != "":
        filters.append(Auction.proaddr.like("%{}%".format(history[0])))
    if history[1] != "":
        filters.append(Auction.numOfBed==history[1])
    if history[2] != "":
        filters.append(Auction.numOfBath==history[2])
    if history[3] != "":
        filters.append(Auction.numOfGarage==history[3])
    if history[4] != "":
        filters.append(Auction.state==history[4])

    filters.append(Auction.finish==0)
    auction_filter = Auction.query.filter(*filters).all()
```

Figure 35. Recommendation based on search history

For the third type of users, we store the last 10 search records of the user in the database, and then access the database according to the most recent search attribute to filter out the real estate that meets the user's search attribute conditions.

When the number of filtered properties is less than three, we will continue to filter again based on the search results of the penultimate user until the filtered properties are greater than or equal to three properties.



```python
if len(result) == 0 or (len(searchhistory) == 0 and search == 1):
    latest_filter = []
    latest_filter.append(Auction.finish==0)

    auction_latest_filter = Auction.query.filter(*latest_filter).all()
    for n in range(len(auction_latest_filter)-1,-1,-1):
        temp = auction_latest_filter[n]
        auc = {}
        auc_dict = temp.__dict__
        auc["id"] = auc_dict["id"]
        if auc_dict["email"] == email or email in eval(auc_dict["bidder"]) or email in eval(auc_dict["observer"]):
            continue
        myid = auc_dict["id"]
        address = auc_dict["proaddr"]
        state = auc_dict["state"]
        zipcode = auc_dict["zipcode"]
        auc["address"] = f"{address}, {state} {zipcode}"
        auc["bedroom"] = auc_dict["numOfBed"]
        auc["bath"] = auc_dict["numOfBath"]
        auc["garage"] = auc_dict["numOfGarage"]
        auc["link"] = f"/auction/id={myid}"
        auc["src"] = eval(auc_dict["pic"])[0]
        auc["status"] = 2
        #if current time is exceeded endtime, set the acution status to finished
        endtime=time.strptime(auc_dict["endtime"],"%Y-%m-%d %H:%M:%S")
        timestamp_endtime = int((time.mktime(endtime)))
        if int(time.time()) > timestamp_endtime:
            temp.finish = 1
            db.session.merge(temp)
            db.session.commit()
        else:
            if len(eval(auc_dict["bidprice"])) < 1:
                auc["bid"] = 0
            else:
                auc["bid"] = eval(auc_dict["bidprice"])[-1][2]
            if auc not in result:
                result.append(auc)
        if len(result) >= 3:
            break
```

Figure 36. Recommend according to the other attributes

The property information will be stored in the result list, and three properties information will be randomly launched in recommendation. If all eligible properties have been checked by the user, the system will display in the recommendation area as the ‘*Sorry! Currently we do not have any recommendation for you*’.

3.2 Special design

3.2.1 Auction state and user type design showed on auction card

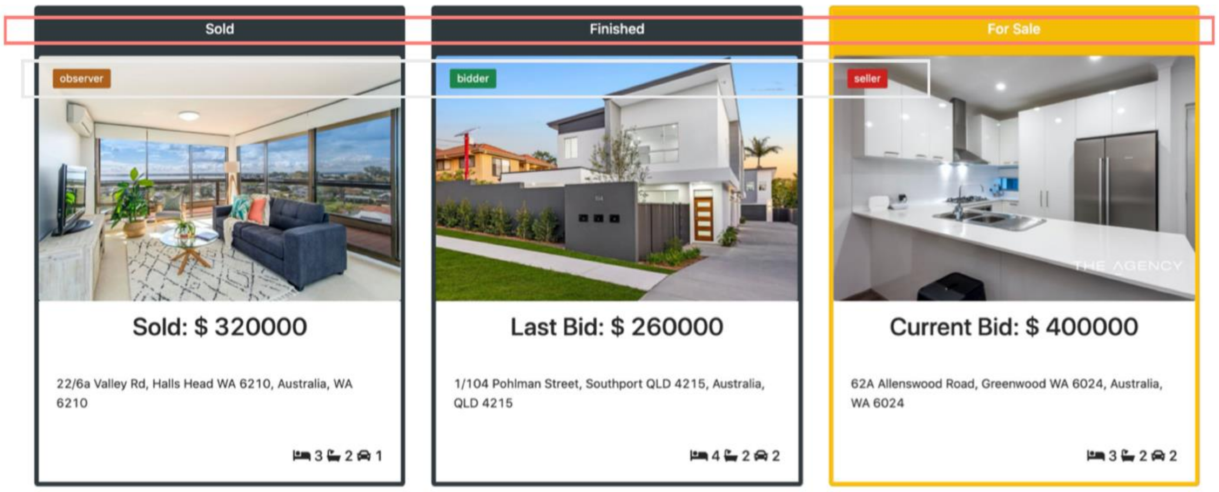
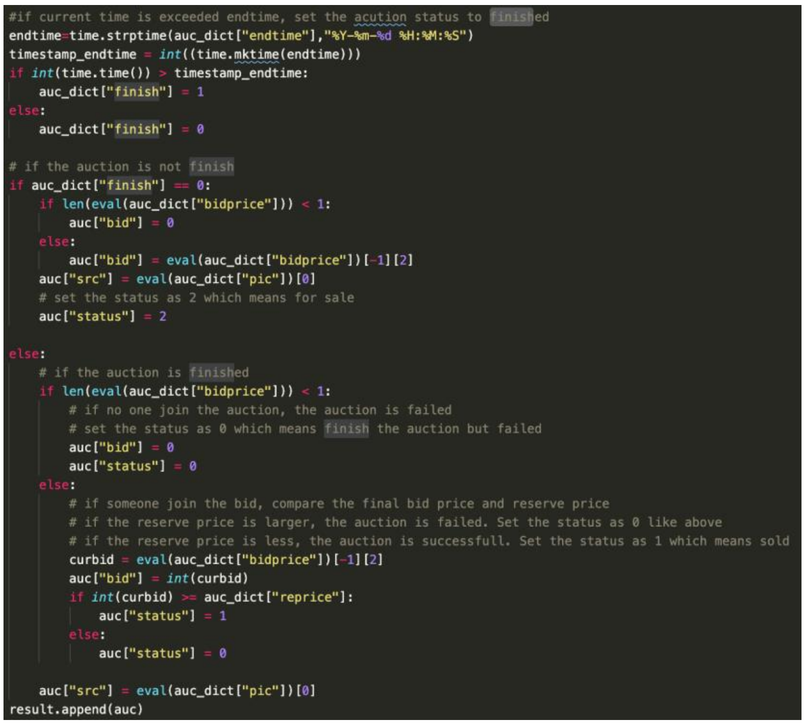
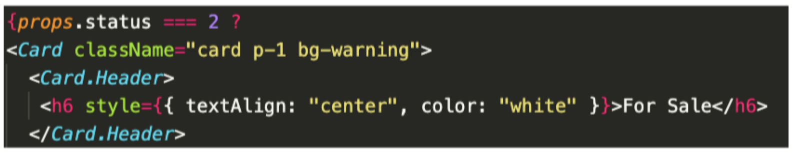

Sold
observer
Sold: $ 320000
22/6a Valley Rd, Halls Head WA 6210, Australia, WA 6210
3 2 1
Finished
bidder
Last Bid: $ 260000
1/104 Pohlman Street, Southport QLD 4215, Australia, QLD 4215
4 2 2
For Sale
seller
THE AGENCY
Current Bid: $ 400000
62A Allenswood Road, Greenwood WA 6024, Australia, WA 6024
3 2 2

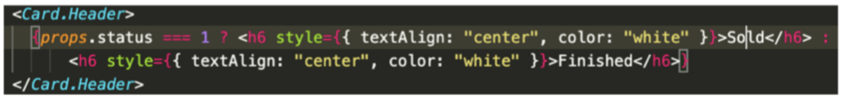
Figure 37. The auction cards in homepage



```
#if current time is exceeded endtime, set the acution status to finished
endtime=time.strptime(auc_dict["endtime"],"%Y-%m-%d %H:%M:%S")
timestamp_endtime = int((time.mktime(endtime)))
if int(time.time()) > timestamp_endtime:
    auc_dict["finish"] = 1
else:
    auc_dict["finish"] = 0

# if the auction is not finish
if auc_dict["finish"] == 0:
    if len(eval(auc_dict["bidprice"])) < 1:
        auc["bid"] = 0
    else:
        auc["bid"] = eval(auc_dict["bidprice"])[-1][2]
    auc["src"] = eval(auc_dict["pic"])[0]
    # set the status as 2 which means for sale
    auc["status"] = 2

else:
    # if the auction is finished
    if len(eval(auc_dict["bidprice"])) < 1:
        # if no one join the auction, the auction is failed
        # set the status as 0 which means finish the auction but failed
        auc["bid"] = 0
        auc["status"] = 0
    else:
        # if someone join the bid, compare the final bid price and reserve price
        # if the reserve price is larger, the auction is failed. Set the status as 0 like above
        # if the reserve price is less, the auction is successfull. Set the status as 1 which means sold
        curbid = eval(auc_dict["bidprice"])[-1][2]
        auc["bid"] = int(curbid)
        if int(curbid) >= auc_dict["reprice"]:
            auc["status"] = 1
        else:
            auc["status"] = 0

    auc["src"] = eval(auc_dict["pic"])[0]
result.append(auc)
```



```
{props.status === 2 ?
<Card className="card p-1 bg-warning">
  <Card.Header>
   <h6 style={{ textAlign: "center", color: "white" }}>For Sale</h6>
  </Card.Header>
```

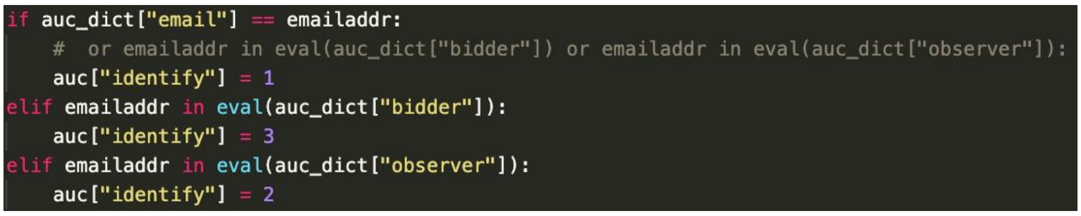


```
<Card.Header>
  {props.status === 1 ? <h6 style={{ textAlign: "center", color: "white" }}>Sold</h6> :
      <h6 style={{ textAlign: "center", color: "white" }}>Finished</h6>}
</Card.Header>
```

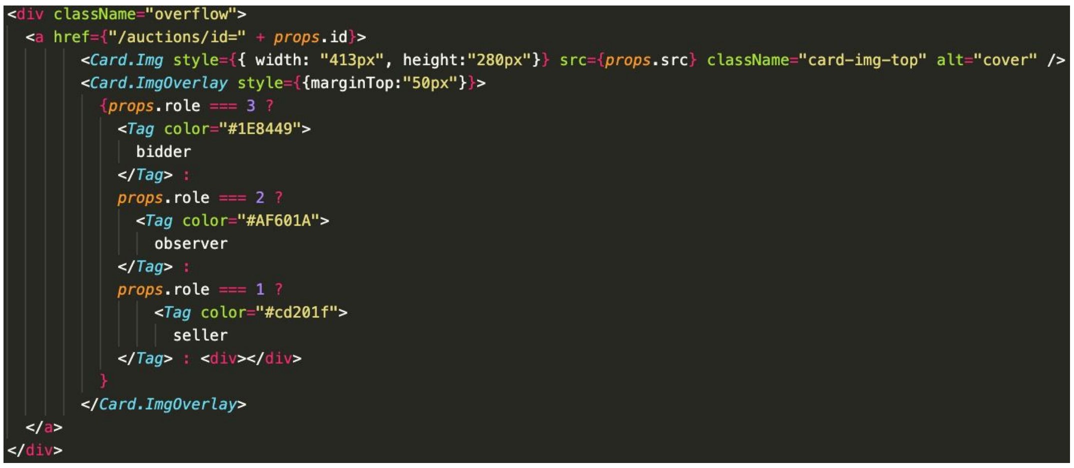
Figure 38. Part of auth.py and AuctionItem.js files related to the auction status on card

The auction has three states: '*sold*', '*finished*' and '*for sale*'. Our team sets the closed auction card with black border, in-progress auction card with yellow border.

In the auth.py file (Figure 38), if the auction closed and the final bid greater or equal to the seller reserved price, the auction state is '*Sold*', set `auc["status"] = 1`. If the auction closed and the final bid less than the seller reserved price, the auction state is '*Finished*', set `auc["status"] = 0` . If the auction in-progress, the auction state is '*For Sale*', set `auc["status"] = 2`. Back-end returns the '*auc*' dictionary to front-end, according to the '*status*' value, front-end config different card header and border for different state auctions.



```
if auc_dict["email"] == emailaddr:
    #  or emailaddr in eval(auc_dict["bidder"]) or emailaddr in eval(auc_dict["observer"]):
    auc["identify"] = 1
elif emailaddr in eval(auc_dict["bidder"]):
    auc["identify"] = 3
elif emailaddr in eval(auc_dict["observer"]):
    auc["identify"] = 2
```



```
<div className="overflow">
  <a href={"/auctions/id=" + props.id}>
      <Card.Img style={{ width: "413px", height:"280px"}} src={props.src} className="card-img-top" alt="cover" />
      <Card.ImgOverlay style={{marginTop:"50px"}}>
        {props.role === 3 ?
          <Tag color="#1E8449">
            bidder
          </Tag> :
          props.role === 2 ?
            <Tag color="#AF601A">
              observer
          </Tag> :
          props.role === 1 ?
              <Tag color="#cd201f">
                seller
          </Tag> : <div></div>
        }
      </Card.ImgOverlay>
  </a>
</div>
```

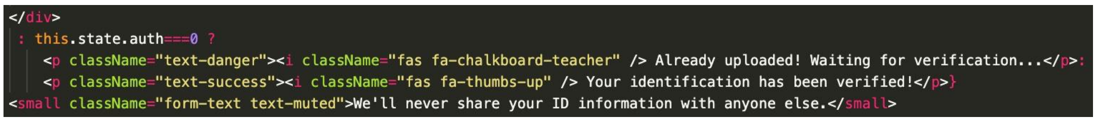
Figure 39. Part of auth.py and AuctionItem.js files related to different user types on card

The user type label on the homepage auction card can be categorized into three types with different colors: '*seller*', '*bidder*' and '*observer*'. In the auth.py file

(Figure 39), if the email address of the current user is same with the email address of the current property poster, which means the current property belongs to the current user, then the back-end sets `auc["identify"] = 1`. If the user's email address is same with the bidder's email address, then the back-end sets `auc["identify"] = 3`. If the user's email address belongs to the observer list, then the back-end sets `auc["identify"] = 2`. If the user is both bidder and observer, cards only show bidder label because the priority of bidder is higher than observer. Back-end returns the '*auc*' dictionary and front-end extracts the '*identity*' value to set different types of labels.

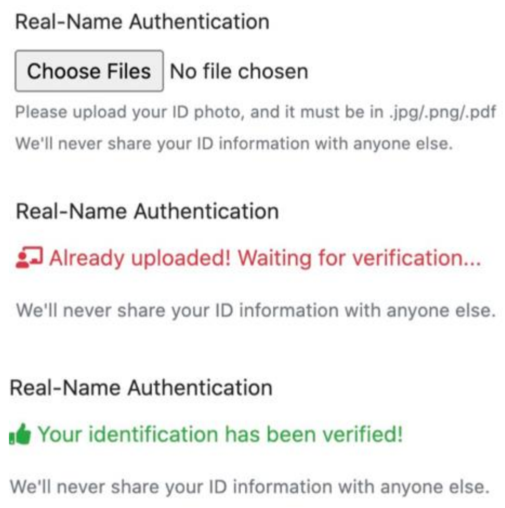
These two special designs make the auction information more clearly and obviously.

3.2.2 Real-name authentication



```
</div>
 : this.state.auth===0 ?
    <p className="text-danger"><i className="fas fa-chalkboard-teacher" /> Already uploaded! Waiting for verification...</p>:
    <p className="text-success"><i className="fas fa-thumbs-up" /> Your identification has been verified!</p>}
<small className="form-text text-muted">We'll never share your ID information with anyone else.</small>
```

Figure 40. Part of Profile.js related to different authentication states



Real-Name Authentication

Choose Files No file chosen

Please upload your ID photo, and it must be in .jpg/.png/.pdf

We'll never share your ID information with anyone else.

Real-Name Authentication

Already uploaded! Waiting for verification...

We'll never share your ID information with anyone else.

Real-Name Authentication

Your identification has been verified!

We'll never share your ID information with anyone else.

Figure 41. Three authentication states

There is a '*checkAuth*' parameter in the user table of our database, the default value of the parameter is 0. If the user passed the authentication which checked by the manual managers, the '*checkAuth*' will change to 1. The users cannot post properties and make bids before their '*checkAuth*' become 1.

4. Third-party functionalities

4.1 Database Layer

SQLite (SQLite Documentation, 2020)

SQLite is the most used database engine in the world which implements a small, self-sufficient, serverless, zero-configuration, transactional SQL database engine. It can support mainstream operating systems such as Windows, Linux and Unix. It is small enough and easy to use for us to arrange it in the laptops during the development process, which is helpful for debugging and system management. We use Navicat 15 as a database tool to create, manage and maintain the database which is very useful for debugging during development.

4.2 Front-end Layer

React.js (React – A JavaScript library for building user interfaces, 2020)

React is a JavaScript library for building user interfaces, which makes it easy to create UIs. It is very efficient to use existing components to form a more complex UI.

React-antd (Components overview - Ant Design, 2020)

Antd is a React UI component library based on Ant Design system, which provides high-quality React components for use. These components have a good style and can be used directly in our UIs or modified to satisfy our requirements.

React-bootstrap (Mark Otto, 2020)

Bootstrap is the most popular front-end framework in the world and react-bootstrap is a rebuild front-end components library in React whose style components depend on bootstrap.

React-file-base64 (Openbase, 2020)

React-file-base64 is a react component to convert files to base64 which is very useful for us to store pdf files and images into the database. This component is based on readAsDataURL function in FileReader object which contains the file string encoded with base64.

React-hook (Introducing Hooks – React, 2020)

React-hook is a method of updating state when class is not applicable which can make react components also have state and implement common state operations between the components. It provides two hook functions called useState and useEffect to setup stack hook. UseState can be used to update state and useEffect can add side effect logic to components such as generating interactions.

Node.js (Node.js, 2020)

Node.js is a platform based on the Chrome JavaScript runtime which can execute JavaScript very fast and has a pretty good performance.

Npm (npm | build amazing things, 2020)

Npm is a package management tool installed with Node.js. It allows users to download and import the third-party packages which is very useful for development.

Fetch (Fetch API, 2020)

Fetch is a web API which provides a JavaScript interface for accessing and manipulating request and response in our project.

4.3 Back-end Layer

Flask (Welcome to Flask — Flask Documentation (1.1.x), 2020)

Flask is a web framework relies on Jinja engine and Werkzeug WSGI suite. Using this framework, we can interact with front-end easily.

SQLAlchemy (SQLAlchemy - The Database Toolkit for Python, 2020)

SQLAlchemy is the most famous ORM framework. It is useful to map the table structure of the relational database to the object which uses session object as the connection with recent database. This automatic conversion will make database operation much easier.

PyEmail (PyEmail, 2020)

PyEmail is a python library to send emails using SMTPLIB library.

Smtplib (smtplib — SMTP protocol client — Python 3.9.0 documentation, 2020)

SMTPLIB provides a very convenient way to send email using SMTP protocol.

5. User documentation

5.1 Installation Guide

The online property sales system is designed to run under Windows, Linux and MacOS system. The installation guide is written for the Linux environment and tested on MacOS and Windows environment.

5.1.1 Environment Setup

The project is based on Python 3.7.3. If you do not have Python3, you can follow the instructions and install Python3.7.3, Pip3 from:

https://www.python.org/downloads/source/.

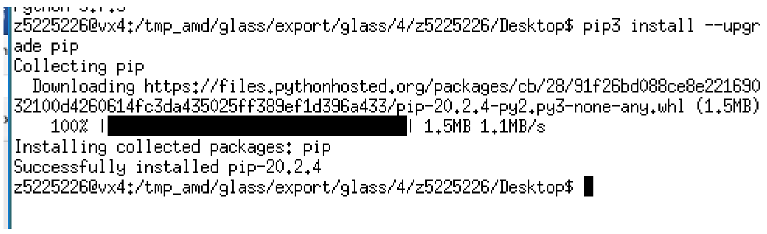
1. Verify the python version using the following command:

   `$ python3 --version`

   Make sure the python version is higher than Python 3.6. Otherwise, the project may not be compatible.

2. Upgrade your pip3 to the latest version using the following command:

   `$pip3 install --upgrade pip`

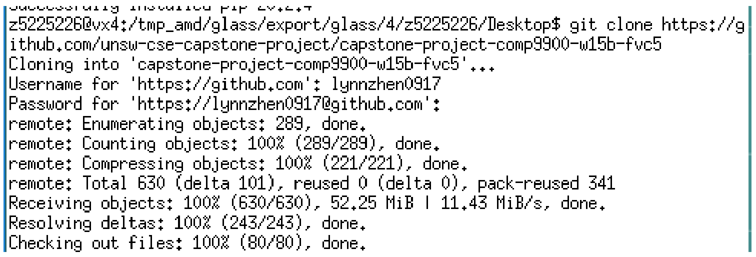


```
z5225226@vx4:/tmp_amd/glass/export/glass/4/z5225226/Desktop$ pip3 install --upgr
ade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/cb/28/91f26bd088ce8e221690
32100d4260614fc3da435025ff389ef1d396a433/pip-20.2.4-py2.py3-none-any.whl (1.5MB)
    100% |████████████████████████████████| 1.5MB 1.1MB/s
Installing collected packages: pip
Successfully installed pip-20.2.4
z5225226@vx4:/tmp_amd/glass/export/glass/4/z5225226/Desktop$ █
```

Figure 42. Upgrade pip3

3. Clone the project from git using the following command:

   `$git clone https://github.com/unsw-cse-capstone-project/capstone-project-comp9900-w15b-fvc5`

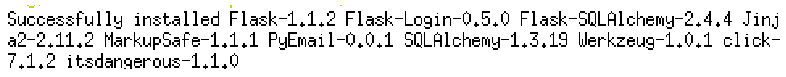


```
z5225226@vx4:/tmp_amd/glass/export/glass/4/z5225226/Desktop$ git clone https://g
ithub.com/unsw-cse-capstone-project/capstone-project-comp9900-w15b-fvc5
Cloning into 'capstone-project-comp9900-w15b-fvc5'...
Username for 'https://github.com': lynnzhen0917
Password for 'https://lynnzhen0917@github.com':
remote: Enumerating objects: 289, done.
remote: Counting objects: 100% (289/289), done.
remote: Compressing objects: 100% (221/221), done.
remote: Total 630 (delta 101), reused 0 (delta 0), pack-reused 341
Receiving objects: 100% (630/630), 52.25 MiB | 11.43 MiB/s, done.
Resolving deltas: 100% (243/243), done.
Checking out files: 100% (80/80), done.
```

Figure 43. Clone the project from git

4. Enter the back-end directory and install all the requirements using the following command:

```
$cd capstone-project-comp9900-w15b-fvc5/onlinePropertySale/backend
$pip3 install -r requirements.txt
```



```
Successfully installed Flask-1.1.2 Flask-Login-0.5.0 Flask-SQLAlchemy-2.4.4 Jinj
a2-2.11.2 MarkupSafe-1.1.1 PyEmail-0.0.1 SQLAlchemy-1.3.19 Werkzeug-1.0.1 click-
7.1.2 itsdangerous-1.1.0
```

Figure 44. Install back-end requirements

5. Enter the front-end directory and install npm using the following command:

```
$cd capstone-project-comp9900-w15b-fvc5/onlinePropertySale/frontend
$npm install
```

The project is tested on the version of npm 6.14.8 and node v12.19.0.

5.1.2 Configure and Run the Software

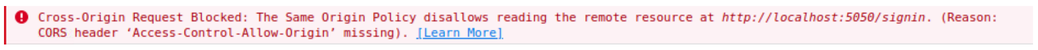
1. After setting up the back-end and front-end environment, you can run the system using the following command:

```
$cd capstone-project-comp9900-w15b-fvc5/onlinePropertySale/backend
$python3 app.py
```

2. After the back-end service is running, start the front-end service using the following command:

```
$cd capstone-project-comp9900-w15b-fvc5/onlinePropertySale/frontend
$npm start
```

3. After running back-end and front-end service, there might be a 'Cross-Origin Request Blocked' problem on your default web browser.



```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://localhost:5050/signin. (Reason:
CORS header 'Access-Control-Allow-Origin' missing). [Learn More]
```

Figure 45. CORS error

Therefore, you need to download an add-on called 'Allow CORS: Access-Control-Allow-Origin' on your web browser (Figure 32) and turn it on.

5.2 Use the system and functionalities

Unlogged-in visitors

A visitor can search the properties with their interested features, they can browse the auction detail page by clicking the auction cards on the homepage. If the visitor wants to bid, observe or post a property, he has to register and log in.

Sign Up

A visitor can go to the ‘*register*’ page by clicking the ‘*Sign Up’* button on the unlogged-in Navbar or clicking the sentence ‘*Do not have an account? Register!*’ in the ‘*signin*’ page. After filling the required information and click the checkbox, the visitor can successfully sign up a new account.

Sign In

After successfully registered as a user, a user can click the ‘*Sign In*’ button and the website will redirect to the ‘*signin*’ page. A user can successfully log in after entering the correct email address and password.

Update Profile

If the user wants to post a property or join as bidder, he has to complete the personal profile. After clicking the username button and choosing the ‘*profile'*, a user can complete and update his profile, including the real-name authentication and the payment details.

Post New Property

Authenticated users can post properties after passing the real-name authentication verification. On the homepage, click the ‘*seller*’ drop-down button on the Navbar, choose the ‘*Post new property*’, users can go to the ‘*newpost*’ page. After completing all the required fields, a new property is posted on the website.

Search Auctions on Homepage

On the homepage, users can search auctions by seven filters (address, auction starting/closing date, number of bedroom/bathroom/garage and state). After choosing their interested attribute and clicking the ‘*Search*’ button, users can see the corresponding auction cards shown under the search box. Users can go to the auction details page by clicking the auction cards.

Auction Details

Each auction will have its auction details page, users can choose to become a bidder or observer on this page. The auction details page shows not only the property but also the auction details, such as the number of bidders/observers. If the user is the seller, bidder or observer of the current auction, he can see the bid history.

User Dashboard

For the seller, bidder and observer, they all have the corresponding dashboard which consists of the current and completed auction cards. The dashboard of the bidder will show the bidder's last bid below the auction card, and the dashboard of the observer has a '*Cancel Observe*' button on the auction card.

Recommendation

Users can see the recommendation of the auction on the homepage when they browse the website. The recommendation is based on the user's latest 10 search history.

Log Out

The user can log out the current account by clicking the '*Logout*' button in the username's drop-down box.

Reference

Sqlite.org. (2020). Sqlite Documentation. Retrieved from https://www.sqlite.org/docs.html

Reactjs.org. (2020). React – A Javascript Library For Building User Interfaces. Retrieved from https://reactjs.org/

Ant.design. (2020). Components overview - Ant Design. Retrieved from https://ant.design/components/overview-/

Mark Otto, a. (2020). Modal. V4.bootcss.com. Retrieved from https://v4.bootcss.com/docs/components/modal/

Openbase.io. (2020). Openbase. Retrieved from https://openbase.io/js/react-file-base64

Reactjs.org. (2020). Introducing Hooks – React. Retrieved from https://reactjs.org/docs/hooks-intro.html

Node.js. (2020). Node.Js. Retrieved from https://nodejs.org/en/

Npmjs.com. (2020). Npm | Build Amazing Things. Retrieved from https://www.npmjs.com

MDN Web Docs. (2020). Fetch API. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Flask.palletsprojects.com. (2020). Welcome To Flask — Flask Documentation (1.1.X*)*. Retrieved from https://flask.palletsprojects.com/en/1.1.x/

Sqlalchemy.org. (2020). Sqlalchemy - The Database Toolkit For Python. Retrieved from https://www.sqlalchemy.org/

PyPI. (2020). Pyemail. Retrieved from https://pypi.org/project/PyEmail/

Docs.python.org. (2020). Smtplib — SMTP Protocol Client — Python 3.9.0 Documentation. Retrieved from https://docs.python.org/3/library/smtplib.html