Zhuhai Property Management System

Group L

Yiting Mao 2130026105

Meng Xu 2130026169

Hanlin Wang 2130026135

Hailin Zhong 2130026215

Submitted: May 20, 2023

Contents

I. Project description

a) The purpose of the project

Real estate is one of the pillar industries of China's national economy. Therefore, in our project, we establish a Zhuhai Property Management System to help users and agents to trade properties. Traditional property trading usually takes place offline, which can be inconvenient and time-consuming. Our project provides an online solution that streamlines the processes involved in inquiring about properties, submitting a home purchase application, and managing agents and users.

b) The difficulty of the problem

The difficulty of managing property properties lies in the vast amount of information that needs to be collected and organized effectively. It needs the procedure of the crawler to collect adequate and authentic data. Meanwhile, the back end should organize these data well, and the front end should show the data efficiently.

What's more, clearly articulating the relationship between the agents, the users, and the admins is also a challenge. Not only do we have to design a logical ER diagram, but the front end also needs to implement three different login interfaces.

c) The abstraction of the problem

The abstraction of the problem in the project involves creating a comprehensive software system that integrates all aspects of property management and trading. To achieve this abstraction, the project will require the development of various modules, such as a subdivision information module, a house information module, a user and agent management module, a submit application module, and so on. These modules will work together to create a cohesive system that simplifies the process of managing properties.

d) The goal of the problem

The major goal of the project is to provide a solution that improves the efficiency of property management in Zhuhai. The system aims to simplify the process of trading properties and improve the overall experience for the agents and users. Ultimately, this project aims to provide a modern, user-friendly, and efficient system that can be adopted by people who want to buy or rent houses in Zhuhai.

II. Assumptions

a) We assume the website has already had 60000 users and 60000 agents.

b) We assume every house should be taken charge of by at most one agent, but one agent can take charge of multiple houses.

c) We assume one user can only apply for a specific house once, but he can apply for multiple houses, and different users can apply for the same house.

d) We believe that people who have registered on this site are either users, admins, or agents.

e) We believe that subdivisions are either new or old.

f) We believe that houses on this site are either new or second-hand. ″

g) Second-hand houses have two statuses, which are rent or sale.

h) We believe that houses cannot exist outside of the subdivisions.

i) Every district has houses.

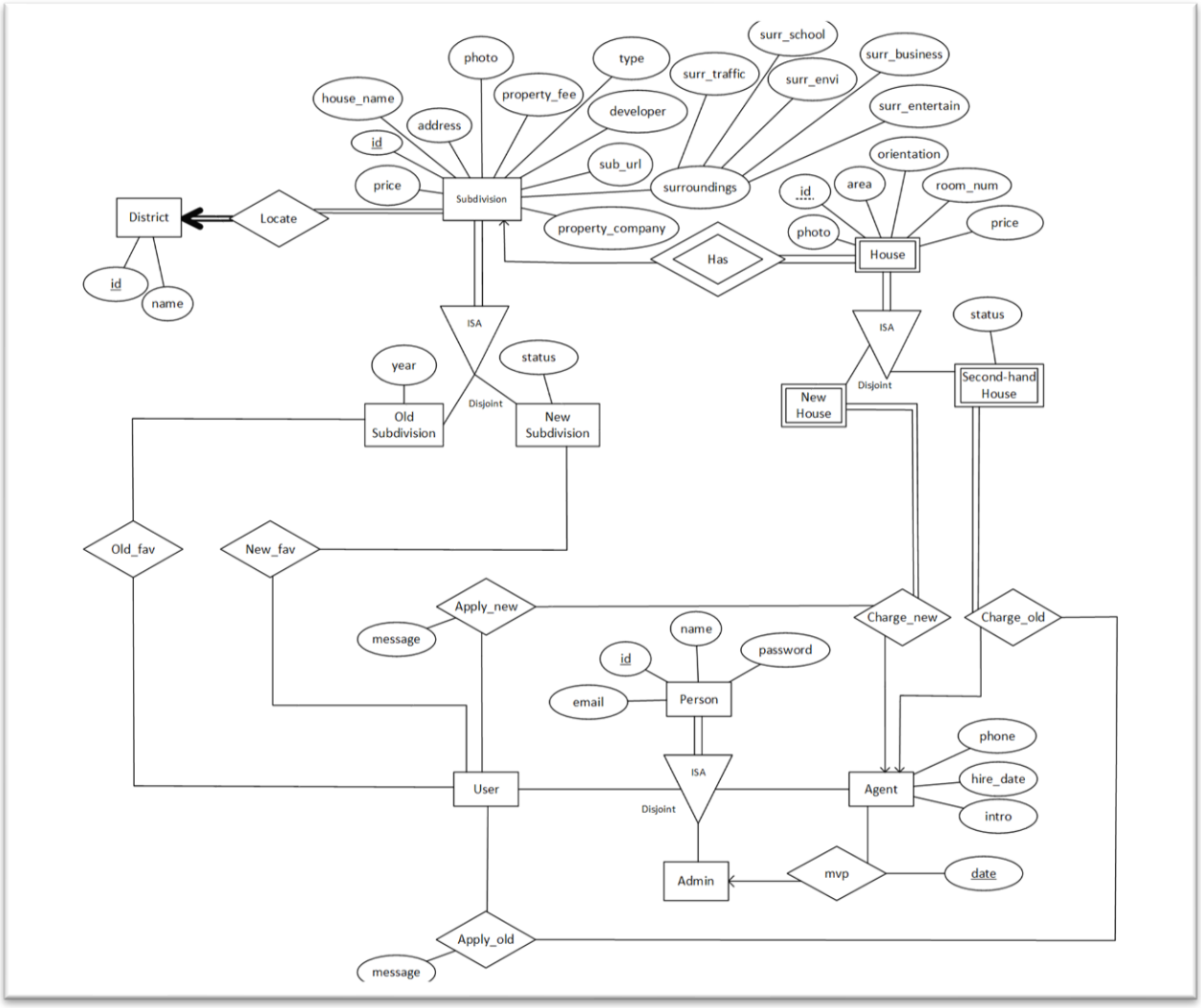
j) We assume one user can only collect the subdivision once.

k) We believe no subdivision has a price with “0”.

l) We believe no houses have a price of “0”, an area with “0”, or several rooms with “0”.

III. Conceptual Design

a) ER diagram



b) Description of the entity set

i. Subdivision

✧ **Introduction:** A subdivision is a parent entity set of a new subdivision and an old subdivision.

Subdivision total participates in two child entities. It contains the common attributes between new subdivisions and old subdivisions.

- ✧ **Primary Key:** "id" is got from ZhengShun.

ii. New Subdivision & Old Subdivision

- ✧ **Introduction:** New Subdivision and Old Subdivision are child entities of Subdivision. Two child entities are disjoint. For the old subdivision, it has a special attribute named "year", and for the new subdivision, it has a special attribute named "status".
- ✧ **Primary Key:** "id", which is inherited from parent entities. The primary key is unique in each child entity set, but there are repeated values between these two entities.

iii. House

- ✧ **Introduction:** House is a weak entity set of the subdivision. It cannot exist alone, which means that its existence depends on the "subdivision" entity set. The house is a parent entity set of new houses and second-hand houses. House total participates in two child entities. It contains the common attributes between new houses and second-hand houses.
- ✧ **Primary Key:** "s_num" represents subdivision ID, which is the primary key. Since it is a weak entity set, it should add the primary key of subdivision. What's more, it also has a discriminator named "h_num".

iv. New House & Second-Hands House

- ✧ **Introduction:** New Houses and Second-Hand Houses are child entities of the House, where new houses only belong to a new subdivision, and old houses belong to an old subdivision. Two child entities are disjoint. For the second-hand house, it has a special attribute named "status", to indicate "rent" or "sale".
- ✧ **Primary Key:** "h_num" and "s_num", where "h_num" is a discriminator and "s_num" is a primary key, are inherited from parent entities. The primary keys are unique as a combination in each child entity set, but they may have repeated values between two child entities.

v. Person

- ✧ **Introduction:** The person entity set includes all people who register on our website. It is a parent entity set of the user, admin, and agent. Person total participates in three child entities. It contains the common attributes between three child entities.
- ✧ **Primary Key:** "id" is unique in each child entity set.

vi. User

- ✧ **Introduction:** The user is the person who registers on our website, searches for information, and makes applications. It is a child entity set of the person, and disjoint with others.
- ✧ **Primary Key:** "id" is the primary key, which is inherited from the Person.

vii. Admin

- ✧ **Introduction:** The admin is the person who takes charge of the website, which means our 4

group members

- ✧ **Primary Key:** “id” is the primary key, which is inherited from the Person.

viii. Agent

- ✧ **Introduction:** The agent is the person who takes charge of the houses. Users can apply for a house from agents
- ✧ **Primary Key:** “id” is the primary key, which is inherited from the Person.

ix. District

- ✧ **Introduction:** The district represents where the subdivision locates.
- ✧ **Primary Key:** “id”, is the primary key.

c) Description of the relationship set

i. Locate

- ✧ **Introduction:** The relationship set “locate” represents where the subdivision locates. The “subdivision” entity set totally participates in the relationship, which means every subdivision should have a location. The “district” entity set also totally participates in the relationship, because during our data-collecting process, our crawler gets data from every district, so it’s impossible that one district has no subdivision. What’s more, the relationship set is one-to-many.

  In this relationship set, we have 14 elements in total, which are the same as the ZhengShun website’s information. Although in this relationship set, we only have 14 elements, it is reasonable and necessary to create a new entity named “district”. If we use the name of the district inside the “subdivision” table to record, it will take up more memory.

- ✧ **Implementation:** We add the primary key of “district” into the “subdivision” as the foreign key.

ii. Has

- ✧ **Introduction:** The relationship set “has”, which is a relationship of the weak entity set, represents that the subdivisions have houses. Specifically, the new subdivision only has new houses, and the old subdivision only has second-hand houses. Since the “house” entity set is a weak entity, it has to be total participation. The “subdivision” partially participates in the relationship. What’s more, the relationship set is one-to-many.
- ✧ **Implementation:** We add the primary key of “subdivision” into the “house” as the foreign key.

iii. New_fav & Old_fav

- ✧ **Introduction:** The relationship sets “new_fav” and “old_fav” indicates the users can collect their favorite subdivision. Since one foreign key cannot relate to two primary keys, we have to separate the favorite sheet. Both “new_fav” and “old_fav” are many-to-many, and the entity sets, users, “new_subdivision”, and “old_subdivision” all partially participate in the relationship.

- ✧ **Implementation:** We create two new tables named “new_fav” and “old_fav”. The primary key of the tables is the primary key of “subdivision” and “users”.

iv. Charge_new & Charge_old

- ✧ **Introduction:** The relationship sets “charge_new” and “charge_old” mean that agents take charge of houses. Both “charge_new” and “charge_old” are one-to-many. The agent partially participates in the relationship. However, “new_house” and “old_house” are totally participation, which means one house must be taken charge of by one agent, but one agent can take charge of multiple houses.
- ✧ **Implementation:** We add the primary key of “agent” into the “house” as the foreign key.

v. Apply_new & Apply_old

- ✧ **Introduction:** The relationship sets “apply_new” and “apply_old” mean that users can apply for a house. The relationship set “apply” is an aggregation to another relationship set named “charge”, because we want to represent that the application that users submit is transformed into a combination of houses and agents to solve. Therefore, both “apply_new” and “apply_old” are many-to-many relationships, and the entity sets, “user”, “charge_new”, and “charge_old” are partially participation.
- ✧ **Implementation:** We create two new tables named “new_fav” and “old_fav”. The primary keys of the relationship sets are the primary key of “subdivision” and “users”.

vi. MVP

- ✧ **Introduction:** The relationship set “MVP” declares that the admin will specify some agents to become MVPs if they perform well. The relationship set is one-to-many. The entity sets “admin” and “agent” partially participate in the relationship, which means not every admin has to assign the MVP, and not every agent can be the MVP. What’s more, it’s possible for one agent to be MVP multiple times.
- ✧ **Implementation:** We create a new table named “mvp”. The primary keys of the relationship set are the primary key of the “admin” and “agent”. Most importantly, the relationship set has a special primary key named “date”, since one agent can be MVP for multiple times, so we have to add the date when they become MVP to identify the difference.

IV.Logical Design

a) Schemas

- ✧ ***Old_Subdivision*** *= (id, district, sub_url, photo, build_year, house_name, address, price, property_company, developer, property_fee, house_type, surr_envi, surr_school, surr_trafic, surr_business, surr_entertain)*
- ✧ ***New_Subdivision*** *= (id, district, sub_url, photo, house_name, house_status, house_type, address, price, developer, surr_trafic, surr_school, surr_envi, surr_business, surr_entertain, peoperty_fee, property_company)*
- ✧ ***New_House*** *= (h_num, s_num, agent_id, photo, room_num, area, price, orientation)*
- ✧ ***Old_House*** *= (h_num, s_num, agent_id, photo, room_num, area, price, status, orientation)*

- ***New_Apply*** *= (user_id, sub_id, house_id, message)*
- ***Old_Apply*** *= (user_id, sub_id, house_id, message)*
- ***New_Fav*** *= (sub_id, user_id)*
- ***Old_Fav*** *= (sub_id, user_id)*
- ***Admin*** *= (id, name, password, email)*
- ***Agents*** *= (id, name, password, email, phone, hire_date, intro)*
- ***District*** *= (id, d_name)*
- ***MVP*** *= (agent_id, admin_id, date)*
- ***Users*** *= (id, name, password, email)*

b) Functional dependencies

- **Old_Subdivision:**

$$F = \{id \rightarrow district, sub\_url, photo, house\_name, house\_status, house\_type, address, price, developer, surr\_traffic, surr\_school, surr\_envi, surr\_business, surr\_entertain, property\_fee, property\_company\}$$

- **New_Subdivision:**

$$F = \{id \rightarrow district, sub\_url, photo, house\_name, house\_status, house\_type, address, price, developer, surr\_traffic, surr\_school, surr\_envi, surr\_business, surr\_entertain, property\_fee, property\_company\}$$

- **New_House:**

$$F = \{h\_num, s\_num \rightarrow agent\_id, photo, room\_num, area, price, orientation\}$$

- **Old_House:**

$$F = \{h\_num, s\_num \rightarrow agent\_id, photo, room\_num, area, price, status, orientation\}$$

- **New_Apply:** $F = \{user\_id, sub\_id, house\_id \rightarrow message\}$
- **Old_Apply:** $F = \{user\_id, sub\_id, house\_id \rightarrow message\}$
- **New_Fav:** $F = \{sub\_id, user\_id \rightarrow sub\_id, user\_id\}$
- **Old_Fav:** $F = \{sub\_id, user\_id \rightarrow sub\_id, user\_id\}$
- **Admin:** $F = \{id \rightarrow name, password, email\}$
- **Agents:** $F = \{id \rightarrow name, password, email, phone, hire\_date, intro\}$
- **District:** $F = \{id \rightarrow d\_name\}$
- **MVP:** $F = \{admin\_id, agent\_id, date \rightarrow admin\_id, agent\_id, date\}$
- **Users:** $F = \{id \rightarrow name, id \rightarrow password, id \rightarrow email\}$

c) Explanation:

Our schemas are all in normal. First of all, they follow the first normal form, because every attribute is atomic. What's more, every non-primary key attribute in each schema completely dependent on the

entire candidate key, and non-primary key attributes depend only on candidate keys and not on other non-primary attributes.

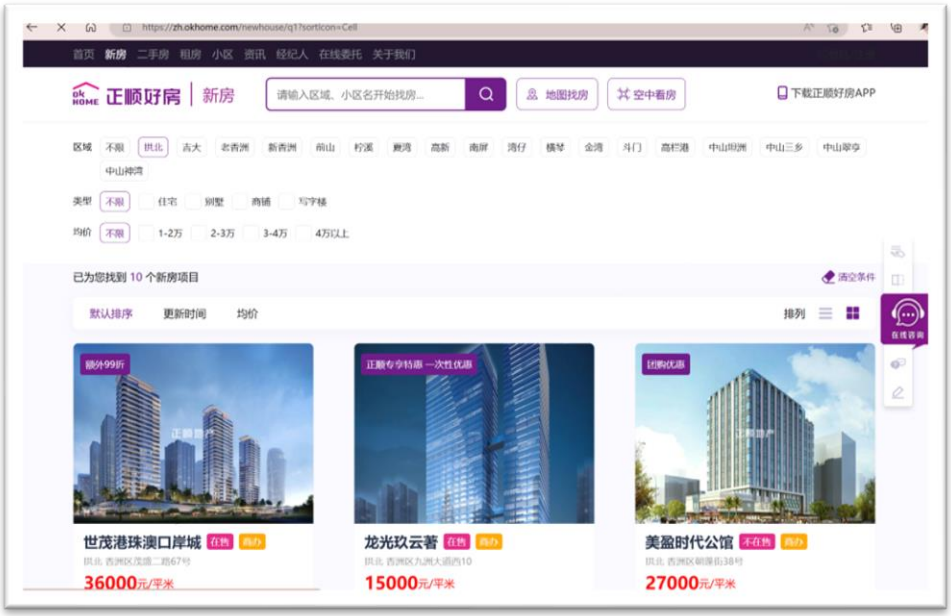
V. Data Description

a) Data acquired by the crawler

In our project, the data related to subdivisions and houses in Zhuhai are all authentic, and by using the crawler we got them from ZhengShun Property 珠海房产网-珠海二手房-珠海租房-珠海新房-珠海买房-正顺好房 (okhome.com).

i. Crawler for new subdivision information

On this web page, we got the following data for a new subdivision in Zhuhai *(to know more about the process, you can check the function named "new_sub_info()" in the file "crawler.ipynb")* :

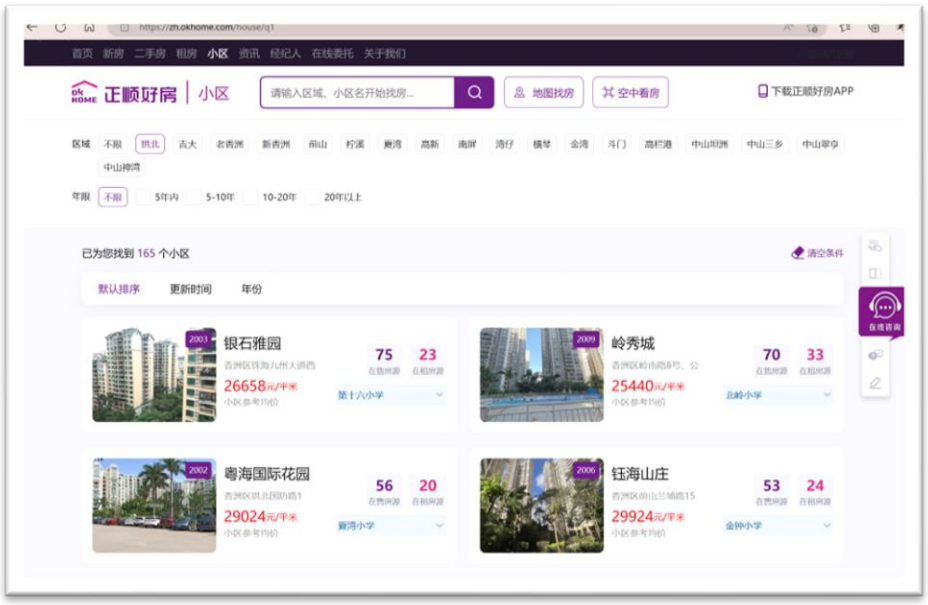


- ✧ **"sub_url"**: This is the URL from ZhengShun's website that can visit the sub-page related to this new subdivision for more detailed information.
- ✧ **"photo"**: The photo of the new subdivision.
- ✧ **"house_name"**: The name of the new subdivision.
- ✧ **"house_status"**: The status of the subdivision, "on-sale" or "not on-sale".
- ✧ **"house_type"**: The type of the subdivision, which can be "commercial", "residential", "villa", or "stores".
- ✧ **"address"**: The address of the new subdivision.
- ✧ **"price"**: The average price which is given by ZhengShun of the new subdivision.
- ✧ **"district"**: The district that the subdivision belongs to.
- ✧ **"id"**: The id is extracted from "sub_url".

You can refer to the document named "new_house_info.csv" to check the data.

ii. Crawler for old subdivision information

On this web page, we got the data for an old subdivision in Zhuhai. It is very similar to the new subdivision's data, except it doesn't have "house_status" and "house_type". What's more, it adds a new attribute named "build_year" *(to know more about the process, you can check the function named "old_sub_info()" in the file "crawler.ipynb")* :

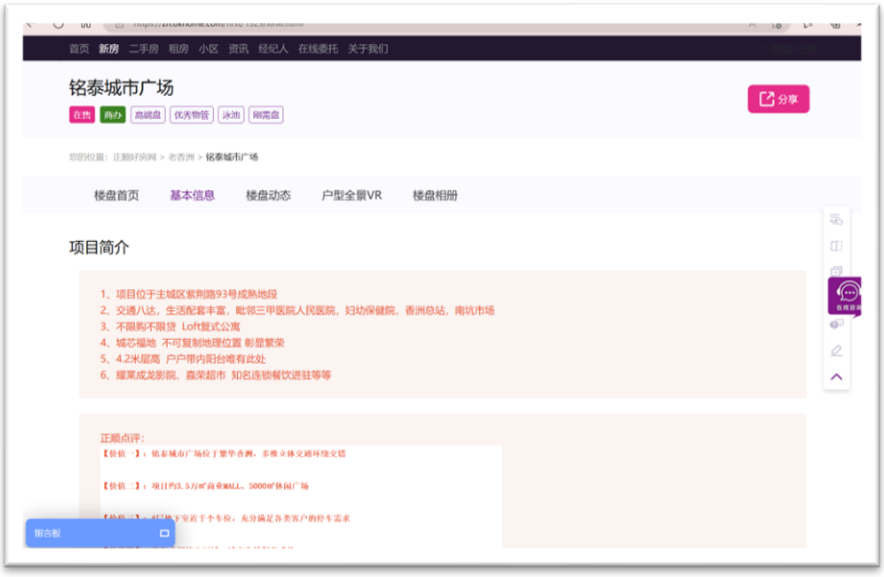


✧ **"build_year"**: Since they are old subdivisions, knowing the building year of the subdivision is crucial.

You can refer to the document named "old_house_info.csv" to check the data.

iii. Crawler for new subdivision detail information

On this web page, we got the detailed information for a new subdivision in Zhuhai *(to know more about the process, you can check the function named "new_sub_detail()" in the file "crawler.ipynb")* :

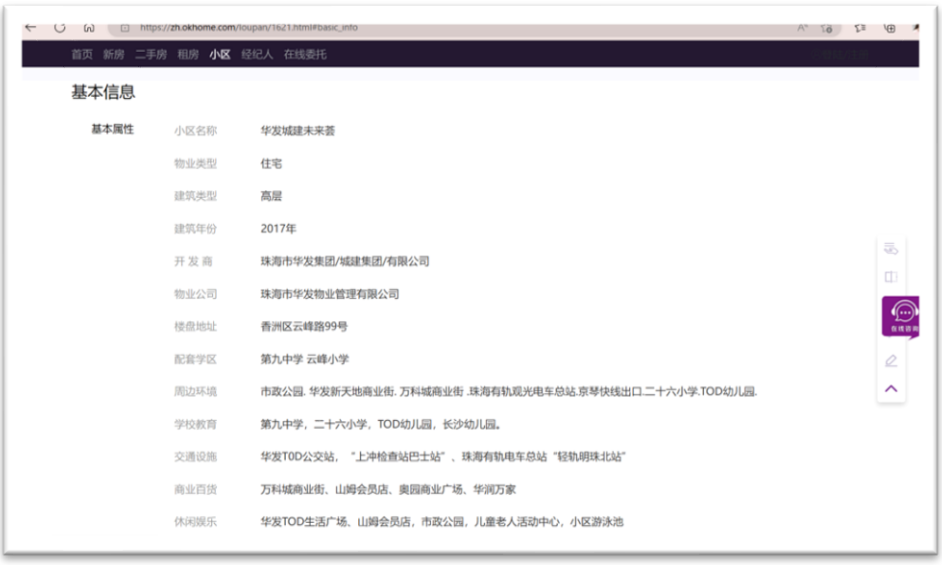


- ✧ **“developer”**: The developer of one new subdivision.
- ✧ **“sur_traffic”**: The traffic situation around the new subdivision.
- ✧ **“sur_school”**: The school around the new subdivision.
- ✧ **“sur_environment”**: The environment around the new subdivision.
- ✧ **“sur_business”**: The business situation around the new subdivision.
- ✧ **“sur_entertain”**: The entertainment around the new subdivision.
- ✧ **“peoperty_fee”**: The property fee of the new subdivision.
- ✧ **“property_company”**: The property company of the new subdivision.

You can refer to the document named “new_house_detail.csv” to check the data.

iv. Crawler for old subdivision detail information

On this web page, we got detailed information about an old subdivision in Zhuhai. It is very similar to the new subdivision’s data, except it adds a new attribute named “house_type” *(to know more about the process, you can check the function named “old_sub_detail()” in the file “crawler.ipynb”)* :

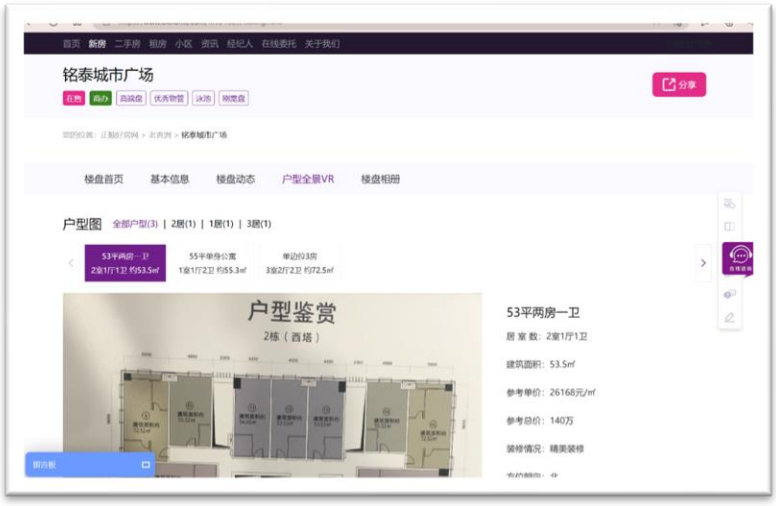


- ✧ **“house_type”**: The type of the subdivision, which can be “commercial”, “residential”, “villa”, or “stores”.

You can refer to the document named “old_house_detail.csv” to check the data.

v. Crawler for the new houses of a related new subdivision

On this web page, we got the information about houses for a new subdivision in Zhuhai *(to know more about the process, you can check the function named “new_house_type()” in the file “crawler.ipynb”)* :

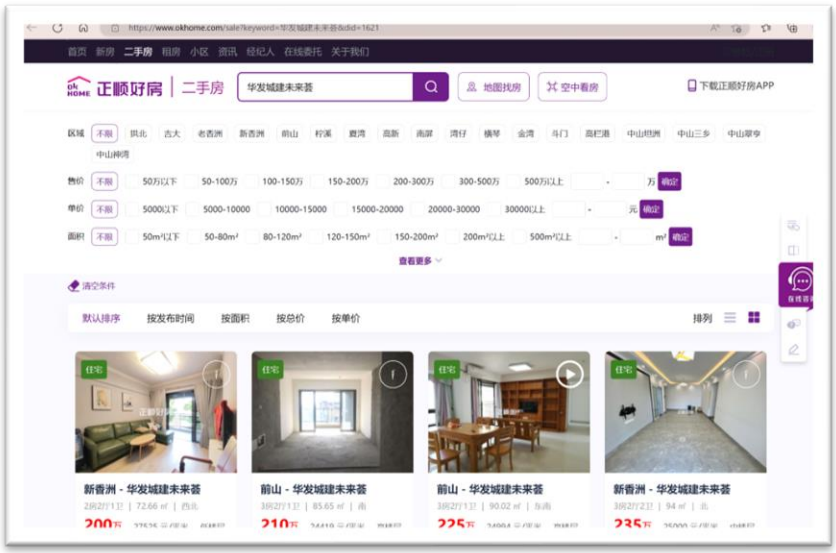


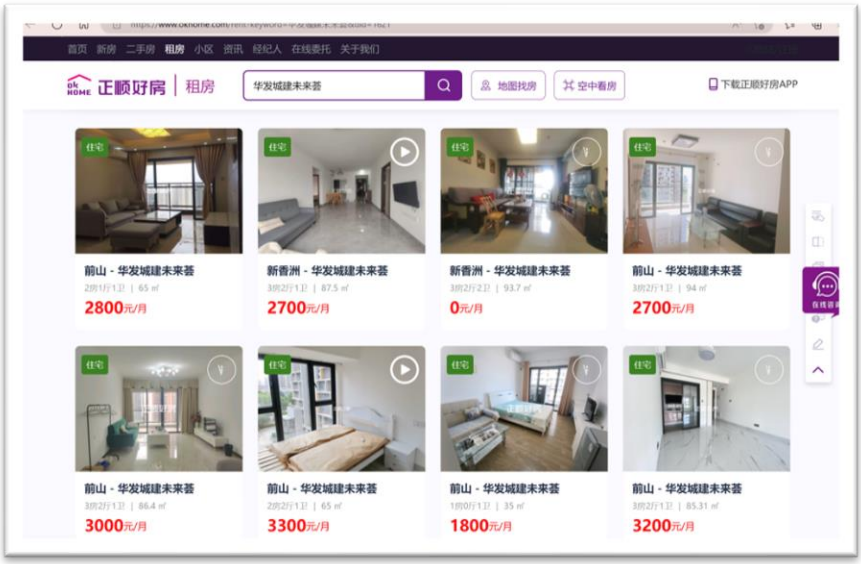
- ✧ **“photo”**: The photo of the new house.
- ✧ **“room_num”**: The number of rooms in the new house.
- ✧ **“area”**: The area of the new house.
- ✧ **“price”**: The price of the new house.
- ✧ **“orientation”**: The orientation of the house.
- ✧ **“h_num”**: The id of the house, generated by ourselves.
- ✧ **“s_num”**: The id of the new subdivision of the new house, which can be used to connect two tables.

You can refer to the document named “new_house_type_info.csv” to check the data.

vi. Crawler for the old houses of a related old subdivision

On these web pages, we got information about houses for an old subdivision in Zhuhai. It is very similar to the new house’s data, except it adds a new attribute named “status” *(to know more about the process, you can check the function named “old_house_type()” in the file “crawler.ipynb”)* :





- ✧ **"status"**: This is used to identify whether it is "rent" or "sale". In our dataset, "0" is for "sale" and "1" is for "rent".
- ✧ **"orientation"**: The data orientation for the old house is different from the new house, where new houses' data are authentic, and for old houses, they are randomly generated, due to the complexity of the crawler.

You can refer to the document named "old_house_type_info.csv" to check the data.

b) Data generated by the "Faker" packet of Python

i. Random users

By using the "Faker" packet of Python, we generate 60000 users' information for our database *(to know more about the process, you can check the function named "random_users()" in the file "fake_data_generate.ipynb")*. The attributes are as follows:

- ✧ **"id"**: The id of users is the only key used to identify different users.
- ✧ **"name"**: The name of the users.
- ✧ **"password"**: The password of the users, which is used to log in to the website.
- ✧ **"email"**: The email of the user.

You can refer to the document named "users.txt" to check the data.

ii. Random agents

By using the "Faker" packet of Python, we generate 60000 agents' information for our database *(to know more about the process, you can check the function named "random_agents()" in the file "fake_data_generate.ipynb")*. It is very similar to the users' data, except it adds some new attributes. The new attributes are as follows:

- ✧ **"phone"**: The phone of agents.
- ✧ **"hire_date"**: The hire_date of agents.

- ✧ **“intro”**: The introduction of agents. We generate about 200 pieces of introduction by using ChatGPT (*you can refer to the document named “introduction_agents.txt”*), and then randomly assigned them to agents.

  You can refer to the document named “agents.txt” to check the data.

iii. Random favorite sheet

To enable the user collection function on the front end, we generate 20000 favorite records in total for our database *(to know more about the process, you can check the function named “new_random_fav()” and “old_random_fav()” in the file “fake_data_generate.ipynb”)*. Due to our database design, the favorite sheet is divided into two parts, one refers to new houses, and the other one refers to old houses. The attributes are as follows:

- ✧ **“sub_id”**: The id of the subdivision, so we can know which subdivision they like.
- ✧ **“user_id”**: The id of users, so we can know which user has a favorite sheet.

  You can refer to the document named “new_fav.txt” and “old_fav.txt” to check the data.

iv. Random MVP

We allow agents to be MVPs, so we generate 1081 MVP records, which is from 2015 till now, and 10 MVPs for each month, for our database *(to know more about the process, you can check the function named “random_mvp()” in the file “fake_data_generate.ipynb”)*. The attributes are as follows:

- ✧ **“agent_id”**: The id of the agent, so we can know who becomes the MVP.
- ✧ **“admin_id”**: The id of the admin, so we can know who assign the agents to become MVP.
- ✧ **“date”**: The date of the agents to become MVPs, including the year and month.

  You can refer to the document named “mvp.txt” to check the data.

v. Random agents for houses

According to our design, each house should be taken charge of one agent, so we randomly select agent ID and add a new column named “agent_id” for the datasets of new houses and old houses *(to know more about the process, you can check the function named “random_new_agent()” and “random_old_agent()” in the file “fake_data_generate.ipynb”)*. The newly added attribute is:

- ✧ **“agent_id”**: The id of the agent, so we can know who takes charge of the house.

  You can refer to the document named “new_house_all.txt” and “old_house_all.txt” to check the newly added data.

vi. Random application

To enable the user can apply for one house on the front end, we generate 60000 apply records in total for our database *(to know more about the process, you can check the function named “new_random_apply()” and “old_random_apply()” in the file “fake_data_generate.ipynb”)*. Due to our database design, the application sheet is divided into two parts, one refers to new houses, and the other one refers to old houses. The attributes are as follows:

- ✧ **“user_id”**: The id of users, so we can know which user has an application.
- ✧ **“sub_id”**: The id of the subdivision, so we can know which subdivision the applied house belongs to.
- ✧ **“house_id”**: The id of the house, so we can know which house users apply.
- ✧ **“message”**: The message of the application. We generate about 200 pieces of messages by using ChatGPT (*you can refer to the document named “message.txt”*), and then randomly assigned them to apply the sheet.

You can refer to the document named “new_apply.txt” and “old_apply.txt” to check the data.

vii. Admin

The information of admins is not random but authentic, which is related to our 4 group members. The attributes of the admin are the same as the users *(to know more about the process, you can check the function named “admin()” in the file “fake_data_generate.ipynb”)*.

```
u1 = {}
u1["id"] = 1
u1["name"] = "Henry King"
u1["pwd"] = 123456
u1["email"] = "r130026135@mail.uic.edu.hk"
admins.append(u1)
```

```
u2 = {}
u2["id"] = 2
u2["name"] = "Meng Xu"
u2["pwd"] = 123456
u2["email"] = "r130026169@mail.uic.edu.hk"
admins.append(u2)
```

```
u3 = {}
u3["id"] = 3
u3["name"] = "Amy Mao"
u3["pwd"] = 123456
u3["email"] = "r130026105@mail.uic.edu.hk"
admins.append(u3)
```

```
u4 = {}
u4["id"] = 4
u4["name"] = "Dylan CLOCK"
u4["pwd"] = 123456
u4["email"] = "r130026215@mail.uic.edu.hk"
admins.append(u4)
```

You can refer to the document named “admin.txt” to check the data.

viii. District

The information about the district is not random but authentic, which is given by the ZhengShun website *(to know more about the process, you can check the function named “generate_district()” in the file “fake_data_generate.ipynb”)*. We only reserve the district that belongs to Zhuhai.

The attributes are as follows:

- ✧ **“id”**: The id of the district.
- ✧ **“d_name”**: The name of the district.

You can refer to the document named “district.txt” to check the data.

c) Data preprocessing

i. Data Splicing

After we crawled the data, we got "new_house_info.csv", "new_house_detail.csv", "old_house_info.csv" and "old_house_detail.csv". We merged the two files separately based on the same columns ('id') to get "new_subdivision.csv" and "old_subdivision.csv".

ii. Data Cleaning

1. We performed data cleaning on new_subdivision.csv and old_subdivision.csv. We checked both files for data where 'price' was 0 or NULL and removed the data.
2. Since after deleting the data in these two files, there is a possibility that the column 's_num' in the old_house.csv and new_house.csv files will have data that was previously deleted, which will affect our subsequent data import. So we made a judgment on whether the column 's_num' in the old_house.csv and new_house.csv data is in the column 'sub_id' in the old_subdivision.csv and new_subdivision.csv.
3. Similarly, we cleaned the data for old_house.csv and new_house.csv files. Firstly, columns ('s_num', 'h_num') as primary keys duplicate, so we have de-duplicated in both files. And we removed the data where 'room_num' was '0 室 0 厅 0 卫' or was NULL and we checked the number of data in column 'area' with $0m^2$ or $0.0m^2$. We find that there is no data with 'area' equal to $0m^2$ or $0.0m^2$ in old_house.csv, and there are 29 data in new_house.csv. So we chose to delete these data. Then we checked the data with 'price' equal to '0 万' or NULL in old_house.csv and found 3 items. And we checked the data with 'price' equal to '0 元' or NULL in new_house.csv and found 10 items.
4. Also, for our generated 'new_apply.csv' and 'new_fav.csv' files, columns ('user_id', 'sub_id', 'house_id') data and columns ('sub_id', 'user_id') as primary keys duplicate, when importing data will report an error, so we have de-duplicated in both files.
5. Finally we reset the index column for all data. But since there are Chinese characters in the data, it will be garbled first. So we re-encoded the data in 'utf-8' and converted them to txt files.

VI. Database Design

a) Create a Database

i. Firstly, we create a database named 'House', and declare 'ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci'('ENGINE=InnoDB': use 'InnoDB' engine, 'DEFAULT CHARSET=utf8': database default encoding is 'utf-8', 'COLLATE utf8_general_ci': Database proofreading rules) when we create the tables.

ii. According to our schemas, we declare the attributes in the table, set the primary keys, and create the foreign key constraints.

- ✧ 'old_subdivision': PRIMARY KEY ('id'), FOREIGN KEY ('district') REFERENCES district('d_id')
- ✧ 'new_subdivision': PRIMARY KEY ('id'), FOREIGN KEY ('district') REFERENCES district('d_id')

- ✧ ‘old_house’: PRIMARY KEY (‘h_num’, ‘s_num’), FOREIGN KEY (‘s_num’) REFERENCES ‘old_subdivision’(‘id’)
- ✧ ‘new_house’: PRIMARY KEY (‘h_num’, ‘s_num’), FOREIGN KEY (‘s_num’) REFERENCES ‘new_subdivision’(‘id’)
- ✧ ‘users’: PRIMARY KEY (‘u_id’)
- ✧ ‘agents’: PRIMARY KEY (‘u_id’)
- ✧ ‘admins’: PRIMARY KEY (‘u_id’)
- ✧ ‘district’: PRIMARY KEY (‘d_id’)
- ✧ ‘old_fav’: PRIMARY KEY (‘sub_id’, ‘u_id’), FOREIGN KEY (‘sub_id’) REFERENCES ‘old_subdivision’(‘id’)
- ✧ ‘new_fav’: PRIMARY KEY (‘sub_id’, ‘u_id’), FOREIGN KEY (‘sub_id’) REFERENCES ‘new_subdivision’(‘id’)
- ✧ ‘old_apply’: PRIMARY KEY (‘user_id’,’house_id’,‘sub_id’)
- ✧ ‘new_apply’: PRIMARY KEY (‘user_id’,’house_id’,‘sub_id’)
- ✧ ‘mvp’: PRIMARY KEY (‘agent_id’,’admin_id’,’date’), FOREIGN KEY (‘admin_id’) REFERENCES ‘admins’(‘u_id’)

b) Import Data

We use the LOAD DATA INFILE method to import the txt files and csv files. For example:

```
-- add data into old_subdivision
LOAD DATA INFILE 'old_subdivision.txt'
INTO TABLE old_subdivision
CHARACTER SET utf8
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;
```

- ✧ **CHARACTER SET**: Here the encoding set of the file is ‘utf8’.
- ✧ **FILES TERMINATED BY**: Separator – ‘,’ between fields.
- ✧ **ENCLOSED BY**: Field values surrounded by ' “ '.
- ✧ **LINES TERMINATED BY**: Specify the start and end characters of each line.
- ✧ **IGNORE 1 ROWS**: Since the first line in files are names of attributes, so here we ignored the first line.

c) Query

The query is related with frontend functions realization, so we use the variables which appear at frontend when querying.

i. Check user name exist or not

```
SELECT u_id FROM users WHERE u_id = user_id;
```

ii. Check email exist or not

```
SELECT email FROM users WHERE u_id = user_id;
```

iii. Create users

```
INSERT INTO users VALUES (user_id, username, password, email);
```

iv. Update users

```
DELETE FROM users WHERE u_id = user_id;
```

v. **Create agents**

```
UPDATE users
SET u_id = user_id, name = username, password = password, email =
email
WHERE u_id = user_id;
```

vi. **Delete agents**

```
INSERT INTO agents VALUES (user_id, username, password, email, phone,
NOW(), NULL);
```

vii. **Update agents**

```
UPDATE agents
SET u_id = user_id, name = username, password = password, email =
email, phone = phone, intro = intro
WHERE u_id = user_id;
```

viii. **Create admin**

```
INSERT INTO admins VALUES (user_id, username, password, email);
```

ix. **Update admin**

```
UPDATE admins
SET u_id = user_id, name = username, password = password, email =
email
WHERE u_id = user_id;
```

x. **Apply table show in user's interface**

```
(SELECT * FROM new_apply WHERE user_id = user_id)
UNION
(SELECT * FROM old_apply WHERE user_id = user_id);
```

xi. **Fav table show in user's interface**

```
(SELECT * FROM new_fav WHERE u_id = user_id)
UNION
(SELECT * FROM old_fav WHERE u_id = user_id);
```

xii. **Users table show in agent's interface**

```
SELECT * FROM users ORDER BY u_id;
```

xiii. **Charged houses show in agent's interface**

```
(SELECT * FROM new_house WHERE agent_id = user_id)
UNION
(SELECT * FROM old_house WHERE agent_id = user_id);
```

xiv. **Related application show in agent's interface**

```
(SELECT * FROM new_apply JOIN new_house ON (house_id = h_num AND
sub_id = s_num) WHERE agent_id = user_id)
UNION
(SELECT * FROM old_apply JOIN old_house ON (house_id = h_num AND
sub_id = s_num) WHERE agent_id = user_id);
```

xv. **Agent table show in admin's interface**

```
SELECT * FROM agents ORDER BY u_id;
```

xvi. **Show admins interface**

```
SELECT * FROM admins ORDER BY u_id;
```

xvii. **User table show in admin's interface**

```
SELECT * FROM users ORDER BY u_id;
```

xviii. MVP table show in admin's interface

```
SELECT * FROM mvp ORDER BY u_id;
```

xix. Delete fav

```
DELETE FROM (
    (SELECT * FROM new_fav WHERE u_id = user_id)
    UNION
    (SELECT * FROM old_fav WHERE u_id = user_id)
) WHERE sub_id = sub_id;
```

xx. Agent deletes house

```
DELETE FROM (
    (SELECT * FROM new_house WHERE agent_id = user_id)
    UNION
    (SELECT * FROM old_house WHERE agent_id = user_id)
) WHERE h_num = house_id;
```

xxi. Admin add MVP

```
INSERT INTO mvp VALUES (user_id, agent_id, NOW());
```

xxii. Admin deletes MVP

```
DELETE FROM mvp WHERE agent_id = agent_id;
```

xxiii. Add application for new house

```
INSERT INTO new_apply VALUES (user_id, sub_id, house_id, msg);
```

xxiv. Check application in new house

```
SELECT user_id, sub_id, house_id
FROM new_apply
WHERE user_id = user_id AND sub_id = sub_id AND house_id = house_id;
```

xxv. Add application for old house

```
INSERT INTO old_apply VALUES (user_id, sub_id, house_id, msg);
```

xxvi. Check application in old house

```
SELECT user_id, sub_id, house_id
FROM old_apply
WHERE user_id = user_id AND sub_id = sub_id AND house_id = house_id;
```

xxvii. Add fav for new subdivision

```
INSERT INTO new_fav VALUES (sub_id, user_id);
```

xxviii. Check fav in new subdivision

```
SELECT * FROM new_fav WHERE sub_id = sub_id AND user_id = user_id;
```

xxix. Add fav for old subdivision

```
INSERT INTO old_fav VALUES (sub_id, user_id);
```

xxx. Check fav in old subdivision

```
SELECT * FROM old_fav WHERE sub_id = sub_id AND user_id = user_id;
```

xxxi. Show all new subdivision

```
SELECT * FROM new_subdivision ORDER BY price;
```

xxxii. Show one new subdivision

```
SELECT * FROM new_subdivision WHERE id = sub_id;
```

xxxiii. Show all old subdivision

```
SELECT * FROM old_subdivision ORDER BY price;
```

xxxiv. Show one old subdivision

```
SELECT * FROM old_subdivision WHERE id = sub_id;
```

xxxv. Show houses for new subdivision

```
SELECT * FROM new_house JOIN new_subdivision ON (s_num = id) WHERE
s_num = sub_id;
```

xxxvi. Show houses for old subdivision

```
SELECT * FROM old_house JOIN old_subdivision ON (s_num = id) WHERE
s_num = sub_id;
```

xxxvii. Show new houses info

```
SELECT * FROM new_house WHERE s_num = sub_id AND h_num = house_id;
```

xxxviii. Show old houses info

```
SELECT * FROM old_house WHERE s_num = sub_id AND h_num = house_id;
```

d) Constraints And Triggers

i. Constraints

✧ **'new_house_price_domain'**: The price of new_house cannot be "0 元" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
ALTER TABLE new_house ADD CONSTRAINT new_house_price_domain CHECK (price <> "0元" AND price <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '1室1厅1卫', '150m²', '0元', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_price_domain` failed for `project`.`new_house`
```

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '1室1厅1卫', '150m²', '0', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_price_domain` failed for `project`.`new_house`
```

✧ **'new_house_area_domain'**: The area of new_house cannot be "$0m^2$" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
ALTER TABLE new_house ADD CONSTRAINT new_house_area_domain CHECK (area <> "0m²" AND area <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '1室1厅1卫', '0m²', '19387元', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_area_domain` failed for `project`.`new_house`
```

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '1室1厅1卫', '0', '19387元', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_area_domain` failed for `project`.`new_house`
```

✧ **'new_house_room_num_domain'**: The room_num of new_house can not be "0 室 0 厅 0 卫" or "0".

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
ALTER TABLE new_house ADD CONSTRAINT new_house_room_num_domain CHECK (room_num <> "0室0厅0卫" AND room_num <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '0室0厅0卫', '132m²', '19387元', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_room_num_domain` failed for `project`.`new_house`
```

```
INSERT INTO new_house
VALUES (1, 11, 25263, NULL, '0', '132m²', '19387元', "南");
```

MySQL said:

```
#4025 - CONSTRAINT `new_house_room_num_domain` failed for `project`.`new_house`
```

If all the requirements are satisfied:

1 row inserted. (Query took 0.0006 seconds.)

```
INSERT INTO new_house VALUES (1, 11, 25263, NULL, '1室1厅1卫', '150.0m²', '19387元', "南");
```

[Edit inline] [Edit] [Create PHP code]

✧ **'old_house_price_domain'**: The price of old_house cannot be "0 元" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)

```
ALTER TABLE old_house ADD CONSTRAINT old_house_price_domain CHECK (price <> "0元" AND price <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO old_house
VALUES (1, 1, 25263, NULL, '1室1厅1卫', '150.0m²', '0元', 1, "南");
```

MySQL said:

```
#4025 - CONSTRAINT `old_house_price_domain` failed for `project`.`old_house`
```

```
INSERT INTO old_house
VALUES (1, 1, 25263, NULL, '1室1厅1卫', '150.0m²', '0', 1, "南");
```

MySQL said:

```
#4025 - CONSTRAINT `old_house_price_domain` failed for `project`.`old_house`
```

✧ **'old_house_area_domain'**: The area of old_house cannot be "$0m^2$" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
ALTER TABLE old_house ADD CONSTRAINT old_house_area_domain CHECK (area <> "0m²" AND area <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO old_house
VALUES (1, 1, 25263, NULL, '1室1厅1卫', '0m²', '1242元', 1, "南");
```

MySQL said:

```
#4025 - CONSTRAINT `old_house_area_domain` failed for `project`.`old_house`
```

```
INSERT INTO old_house
VALUES (1, 1, 25263, NULL, '1室1厅1卫', '0', '1242元', 1, "南");
```

MySQL said:

```
#4025 - CONSTRAINT `old_house_area_domain` failed for `project`.`old_house`
```

✧ **'old_house_room_num_domain'**: The room_num of old_house can not be "0 室 0 厅 0 卫" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
ALTER TABLE old_house ADD CONSTRAINT old_house_room_num_domain CHECK (room_num <> "0室0厅0卫" AND room_num <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO old_house
VALUES (1, 11, 25263, NULL, '0室0厅0卫', '132㎡', '19387元', 1, "南");
```

MySQL said:

#4025 - CONSTRAINT `old_house_room_num_domain` failed for `project`.`old_house`

```
INSERT INTO old_house
VALUES (1, 11, 25263, NULL, '0', '132㎡', '19387元', 1, "南");
```

MySQL said:

#4025 - CONSTRAINT `old_house_room_num_domain` failed for `project`.`old_house`

If all the requirements are satisfied:

1 row inserted. (Query took 0.0004 seconds.)

```
INSERT INTO old_house VALUES (10, 11, 25263, NULL, '1室1厅1卫', '132㎡', '19387元', 1, "南");
```

[Edit inline] [Edit] [Create PHP code]

✧ **'new_subdivision_price_domain'**: The price of new_subdivision can not be "0 元" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
ALTER TABLE new_subdivision ADD CONSTRAINT new_subdivision_price_domain CHECK (price <> "0元" AND price <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO new_subdivision
VALUES (1, 11, NULL, NULL, NULL, NULL, NULL, NULL, '0元', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

MySQL said:

#4025 - CONSTRAINT `new_subdivision_price_domain` failed for `project`.`new_subdivision`

```
INSERT INTO new_subdivision
VALUES (1, 11, NULL, NULL, NULL, NULL, NULL, NULL, '0', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

MySQL said:

#4025 - CONSTRAINT `new_subdivision_price_domain` failed for `project`.`new_subdivision`

1 row inserted. (Query took 0.0004 seconds.)

```
INSERT INTO new_subdivision VALUES (1, 11, NULL, NULL, NULL, NULL, NULL, NULL, '1937元', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

[Edit inline] [Edit] [Create PHP code]

✧ **'old_subdivision_price_domain'**: The price of old_subdivision can not be "0 元" or "0"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

```
ALTER TABLE old_subdivision ADD CONSTRAINT old_subdivision_price_domain CHECK (price <> "0元" AND price <> "0");
```

[Edit inline] [Edit] [Create PHP code]

Example:

```
INSERT INTO old_subdivision
VALUES (0, 11, NULL, NULL, NULL, NULL, NULL, NULL, '0元', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

MySQL said:

#4025 - CONSTRAINT `old_subdivision_price_domain` failed for `project`.`old_subdivision`

```
INSERT INTO old_subdivision
VALUES (0, 11, NULL, NULL, NULL, NULL, NULL, NULL, '0', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

MySQL said:

#4025 - CONSTRAINT `old_subdivision_price_domain` failed for `project`.`old_subdivision`

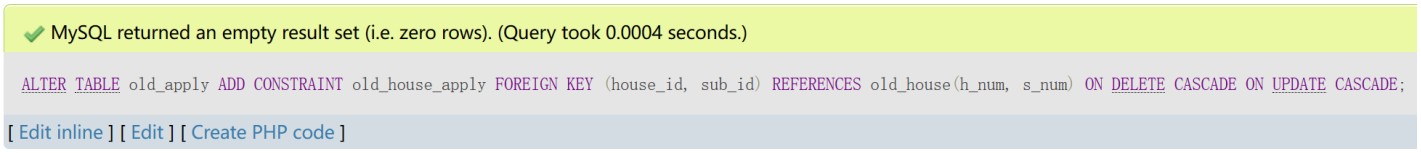
If all the requirements are satisfied:

1 row inserted. (Query took 0.0004 seconds.)

```
INSERT INTO old_subdivision VALUES (0, 11, NULL, NULL, NULL, NULL, NULL, '1983478元', NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

[Edit inline] [Edit] [Create PHP code]

✧ **'old_house_apply'**: If we delete or update one house from 'old_house' then 'old_apply' should also change

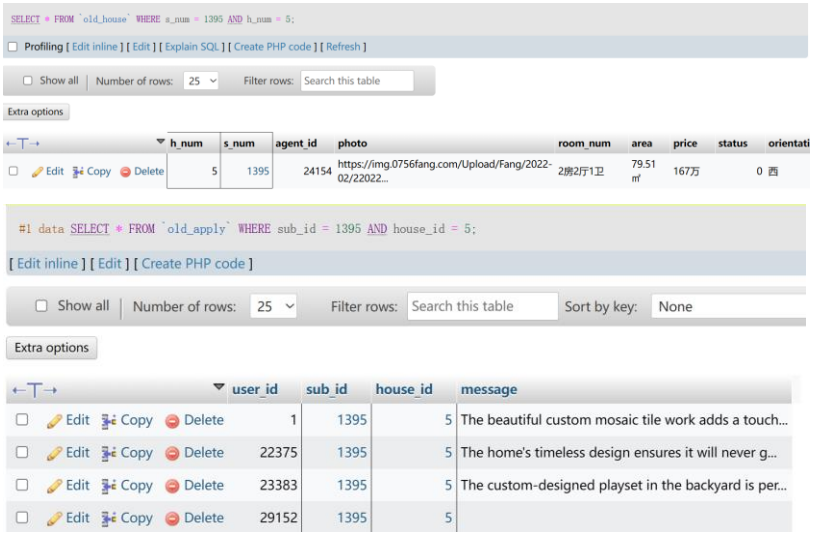


Example

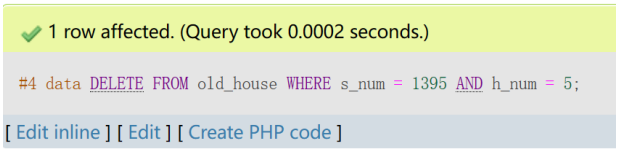
DELETE:

Firstly, we selected the data in tables 'old_house' and 'old_apply' where 's_num' is 1395 and 'h_num' is 5

There are one in 'old_house', five in 'old_apply'



Secondly, we deleted the data in 'old_house'



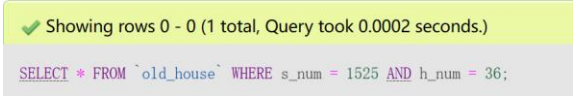
Finally, we selected the data in 'old_apply' where 's_num' is 1395 and 'h_num' is 5, and we found the data were deleted.

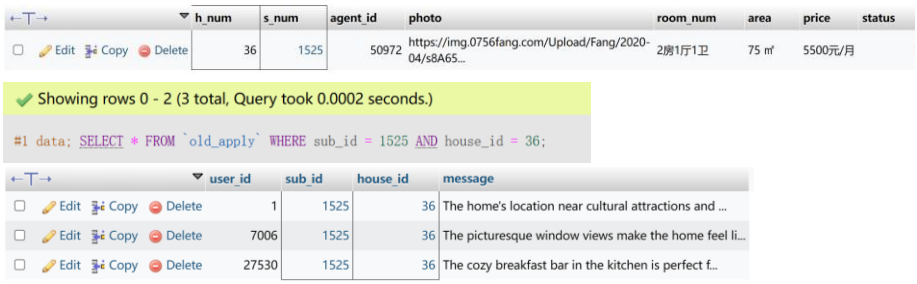
Successfully deleted:

UPDATE:

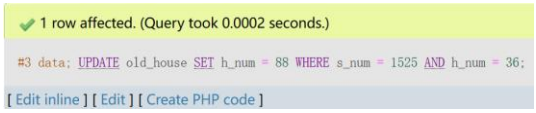
Firstly, we selected the data in tables 'old_house' and 'old_apply' where 's_num' is 1525 and 'h_num' is 36

There are one in 'old_house', three in 'old_apply'

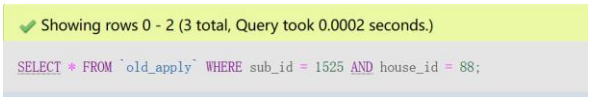




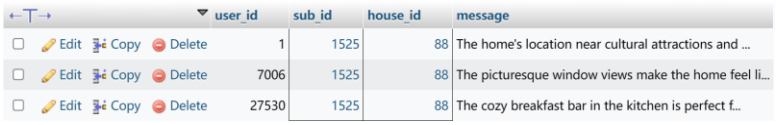
Secondly, we updated the data in 'old_house' and turned the value of 'h_num' from 36 to 88



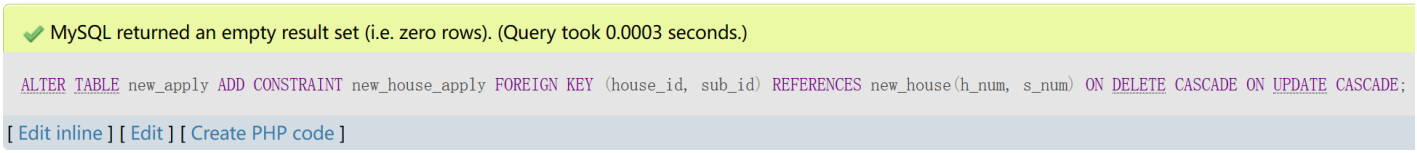
Finally, we selected the data in 'old_apply' where 's_num' is 1525 and 'h_num' is 88, and we found the data were updated.



Successfully changed:



✧ **'new_house_apply'**: If we delete or update one house from 'new_house' then 'new_apply' should also change

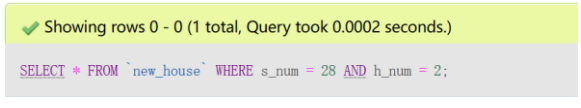


Example:

DELETE:

Firstly, we selected the data in tables 'new_house' and 'new_apply' where 's_num' is 28 and 'h_num' is 2

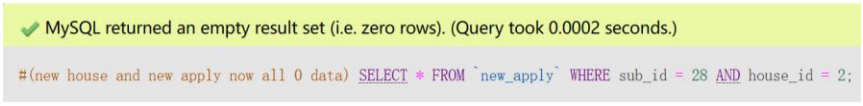
There are one in 'new_house', 36 in 'new_apply'



Secondly, we deleted the data in 'new_house'

Finally, we selected the data in ‘new_apply’ where ‘s_num’ is 28 and ‘h_num’ is 2, and we found the data were deleted.

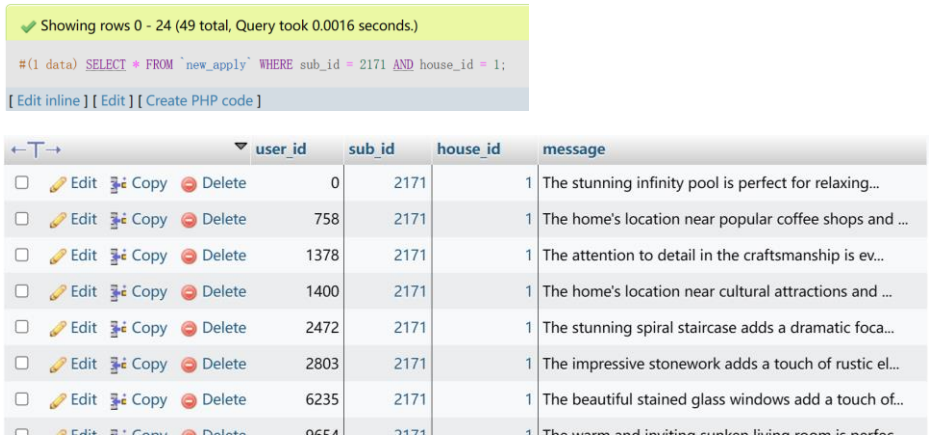
Successfully deleted:



UPDATE:

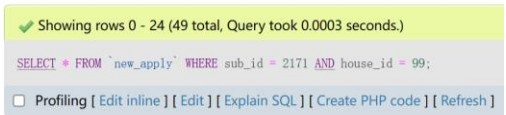
Firstly, we selected the data in tables ‘new_house’ and ‘new_apply’ where ‘s_num’ is 2171 and ‘h_num’ is 1

There are one in ‘new_house’, 49 in ‘new_apply’



Secondly, we updated the data in ‘old_house’ and turned the value of ‘h_num’ from 1 to 99

Finally, we selected the data in ‘old_apply’ where ‘s_num’ is 1525 and ‘h_num’ is 88, and we found the data were updated.



Successfully changed:

	user_id	sub_id	house_id	message
Edit Copy Delete	0	2171	99	The stunning infinity pool is perfect for relaxing...
Edit Copy Delete	758	2171	99	The home's location near popular coffee shops and ...
Edit Copy Delete	1378	2171	99	The attention to detail in the craftsmanship is ev...
Edit Copy Delete	1400	2171	99	The home's location near cultural attractions and ...
Edit Copy Delete	2472	2171	99	The stunning spiral staircase adds a dramatic foca...
Edit Copy Delete	2803	2171	99	The impressive stonework adds a touch of rustic el...
Edit Copy Delete	6235	2171	99	The beautiful stained glass windows add a touch of...

ii. Triggers

✧ **'delete_agent'**: If we delete one agent then 'mvp', 'new_house', and 'old_house' should also change as 'agent_id' changed

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0020 seconds.)

```
CREATE TRIGGER delete_agent BEFORE DELETE ON agents FOR EACH ROW BEGIN IF old.u_id IN ( SELECT DISTINCT agent_id FROM old_house UNION SELECT DISTINCT agent_id FROM new_house ) THEN UPDATE new_house SET agent_id = (SELECT u_id FROM agents ORDER BY rand() LIMIT 1) WHERE new_house.agent_id = old.u_id; UPDATE old_house SET agent_id = (SELECT u_id FROM agents ORDER BY rand() LIMIT 1) WHERE old_house.agent_id = old.u_id; DELETE FROM mvp WHERE agent_id = old.u_id; END IF; END;;
```

[Edit inline] [Edit] [Create PHP code]

Example:

For 'new_house':

Showing rows 0 - 0 (1 total, Query took 0.0010 seconds.)

```
SELECT * FROM `agents` WHERE u_id = 42542;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Showing rows 0 - 0 (1 total, Query took 0.0005 seconds.)

```
-- 1 data SELECT * FROM `new_house` WHERE agent_id = 42542;
```

[Edit inline] [Edit] [Create PHP code]

	h_num	s_num	agent_id	photo	room_num	area	price	orientatio
Edit Copy Delete	0	11	42542	https://img.0756fang.com/Upload/Dictionary/2021-10...	1室1厅1卫	150.0㎡	16000元	南

We deleted the data in 'agents'

1 row affected. (Query took 0.0003 seconds.)

```
-- 1 data DELETE FROM agents WHERE u_id = 42542;
```

[Edit inline] [Edit] [Create PHP code]

But we can't have an agent in 'new_house', so we'll assign him a random agent: here we can see 'agengt_id' of 'new_house' now is 11442

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT * FROM new_house where h_num = 0 AND s_num = 11;
```

	h_num	s_num	agent_id	photo	room_num	area	price	orientation
Edit Copy Delete	0	11	11442	https://img.0756fang.com/Upload/Dictionary/2021-10...	1室1厅1卫	150.0㎡	16000元	南

For 'old_house':

Showing rows 0 - 0 (1 total, Query took 0.0003 seconds.)

```
SELECT * FROM `agents` WHERE u_id = 14997;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Showing rows 0 - 2 (3 total, Query took 0.0063 seconds.)

```
-- 1 data SELECT * FROM `old_house` WHERE agent_id = 14997;
```

[Edit inline] [Edit] [Create PHP code]

	h_num	s_num	agent_id	photo	room_num	area	price	status	orientatio
Edit Copy Delete	0	1	14997	https://img.0756fang.com/Upload/Fang/2019-12/7d74d...	1房0厅0卫	60.3 ㎡	271万	0	北
Edit Copy Delete	6	3557	14997	https://img.0756fang.com/Upload/Fang/2021-04/s9E11...	1房0厅0卫	67 ㎡	5360元/月	1	西南
Edit Copy Delete	7	1280	14997	https://img.0756fang.com/Upload/Fang/2021-10/s173A...	3房2厅2卫	167.04 ㎡	530万	0	西

We deleted the data in 'agents'

1 row affected. (Query took 0.0003 seconds.)

```
-- 3 data DELETE FROM agents WHERE u_id = 14997;
```

[Edit inline] [Edit] [Create PHP code]

But we can't have an agent in 'old_house', so we'll assign him a random agent: here we can see 'agengt_id' of 'old_house' now is 38322(condition: 'h_num' = 0 and 's_num' = 1)

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT * FROM `old_house` WHERE h_num = 0 AND s_num = 1;
```

	h_num	s_num	agent_id	photo	room_num	area	price	status	orientation
Edit Copy Delete	0	1	38322	https://img.0756fang.com/Upload/Fang/2019-12/7d74d...	1房0厅0卫	60.3 ㎡	271万	0	北

The second one is 16925(condition: 'h_num' = 6 and 's_num' = 3557)

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT * FROM `old_house` WHERE h_num = 6 AND s_num = 3557;
```

	h_num	s_num	agent_id	photo	room_num	area	price	status
Edit Copy Delete	6	3557	16925	https://img.0756fang.com/Upload/Fang/2021-04/s9E11...	1房0厅0卫	67 ㎡	5360元/月	1

The second one is 16925(condition: 'h_num' = 6 and 's_num' = 3557)

Showing rows 0 - 0 (1 total, Query took 0.0001 seconds.)

```
SELECT * FROM `old_house` WHERE h_num = 7 AND s_num = 1280;
```

	h_num	s_num	agent_id	photo	room_num	area	price	status
Edit Copy Delete	7	1280	31486	https://img.0756fang.com/Upload/Fang/2021-10/s173A...	3房2厅2卫	167.04 ㎡	530万	0

From the above three cases we can also see that we are randomly assigned agent

✧ **'update_agent'**: If we update one agent then 'mvp', 'new_house', and 'old_house' should also change as 'agent_id' changed

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0023 seconds.)

```
CREATE TRIGGER update_agent BEFORE UPDATE ON agents FOR EACH ROW BEGIN IF old.u_id IN ( SELECT DISTINCT agent_id FROM old_house UNION SELECT DISTINCT agent_id FROM new_house ) THEN UPDATE new_house SET agent_id = new.u_id WHERE new_house.agent_id = old.u_id; UPDATE old_house SET agent_id = new.u_id WHERE old_house.agent_id = old.u_id; UPDATE mvp SET agent_id = new.u_id WHERE mvp.agent_id = old.u_id; END IF; END;;
```

[Edit inline] [Edit] [Create PHP code]

Example:

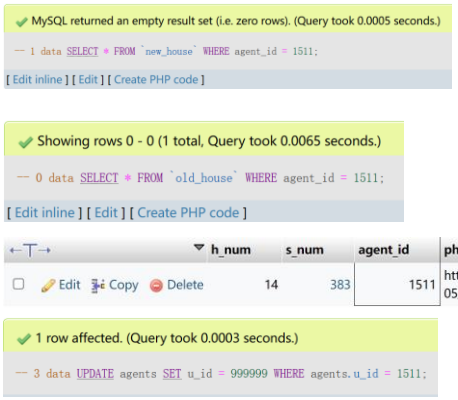
For 'old_house' and 'new_house':

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

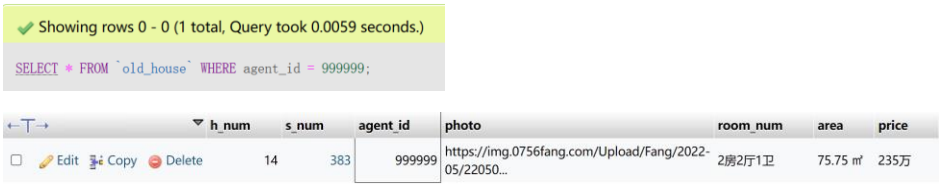
```
SELECT * FROM `agents` WHERE u_id = 1511;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

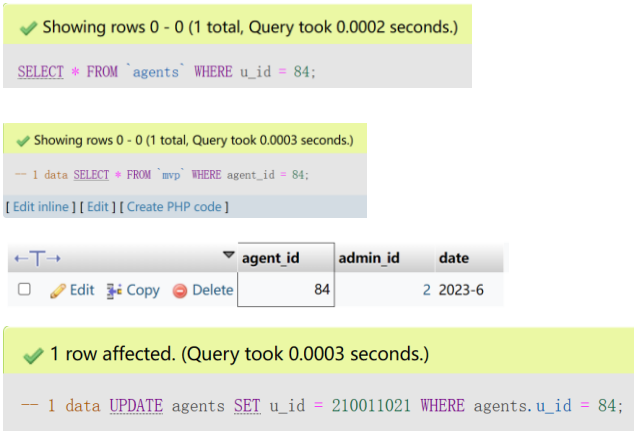
Here there is no ‘agent_id’ = 1511 data in ‘new_house’:



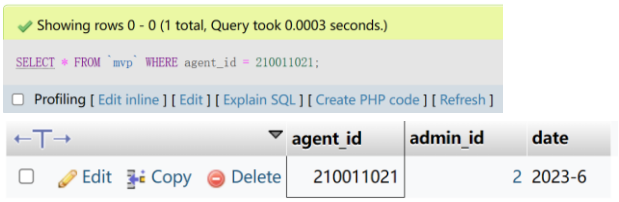
Since there is no data in ‘new_house’, here we only selected the data in ‘old_house’ and the data was updated:



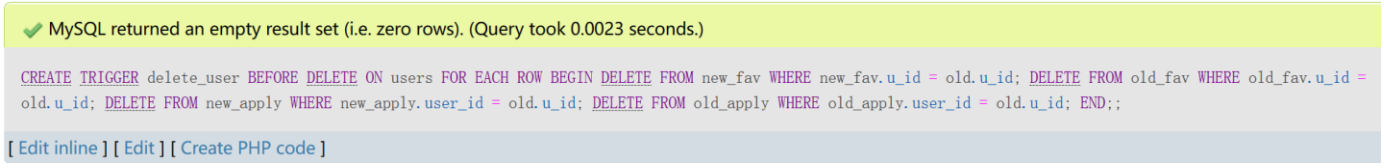
For ‘mvp’:



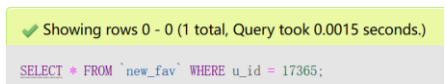
Here the data in ‘mvp’ was also updated.



✧ **‘delete_user’**: If we delete one user then ‘new_fav’, ‘old_fav’, ‘new_apply’, and ‘old_apply’ change as ‘u_id’ changed.



Example:



Before operations - Four tables:

Showing rows 0 - 0 (1 total, Query took 0.0013 seconds.)

```
SELECT * FROM `old_fav` WHERE u_id = 17365;
```

Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT * FROM `new_apply` WHERE user_id = 17365;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Showing rows 0 - 0 (1 total, Query took 0.0003 seconds.)

```
SELECT * FROM `old_apply` WHERE user_id = 17365;
```

1 row affected. (Query took 0.0003 seconds.)

```
DELETE FROM users WHERE u_id = 17365;
```

After operations - four tables:

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```
SELECT * FROM `new_fav` WHERE u_id = 17365;
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```
SELECT * FROM `old_fav` WHERE u_id = 17365;
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
SELECT * FROM `new_apply` WHERE user_id = 17365;
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
SELECT * FROM `old_apply` WHERE user_id = 17365;
```

- ‘**update_user**’: If we update one user then ‘new_fav’, ‘old_fav’, ‘new_apply’, and ‘old_apply’ change as ‘u_id’ changed

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0024 seconds.)

```
CREATE TRIGGER update_user BEFORE update ON users FOR EACH ROW BEGIN UPDATE new_fav SET u_id = new.u_id WHERE new_fav.u_id = old.u_id; UPDATE old_fav SET u_id = new.u_id WHERE old_fav.u_id = old.u_id; UPDATE new_apply SET user_id = new.u_id WHERE new_apply.user_id = old.u_id; UPDATE old_apply SET user_id = new.u_id WHERE old_apply.user_id = old.u_id; END;;
```

Before operations - Four tables:

Showing rows 0 - 0 (1 total, Query took 0.0015 seconds.)

```
SELECT * FROM `new_fav` WHERE u_id = 52195;
```

	sub_id	u_id
Edit Copy Delete	2135	52195

Showing rows 0 - 0 (1 total, Query took 0.0015 seconds.)

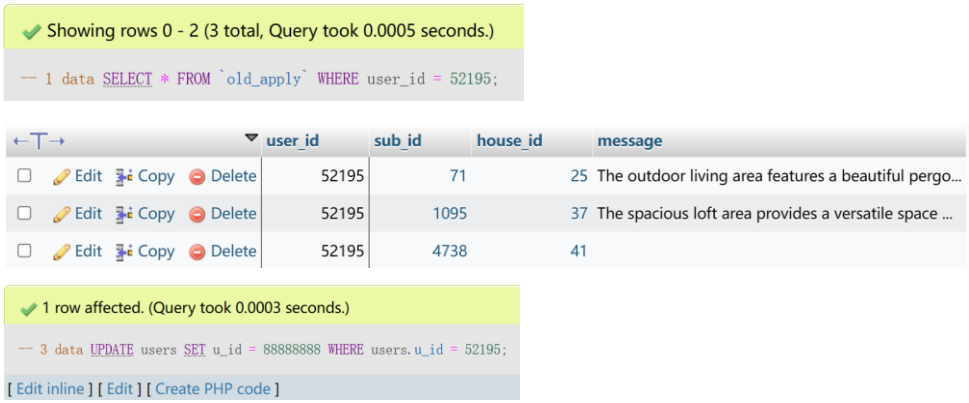
```
-- 1 data SELECT * FROM `old_fav` WHERE u_id = 52195;
```

	sub_id	u_id
Edit Copy Delete	59	52195

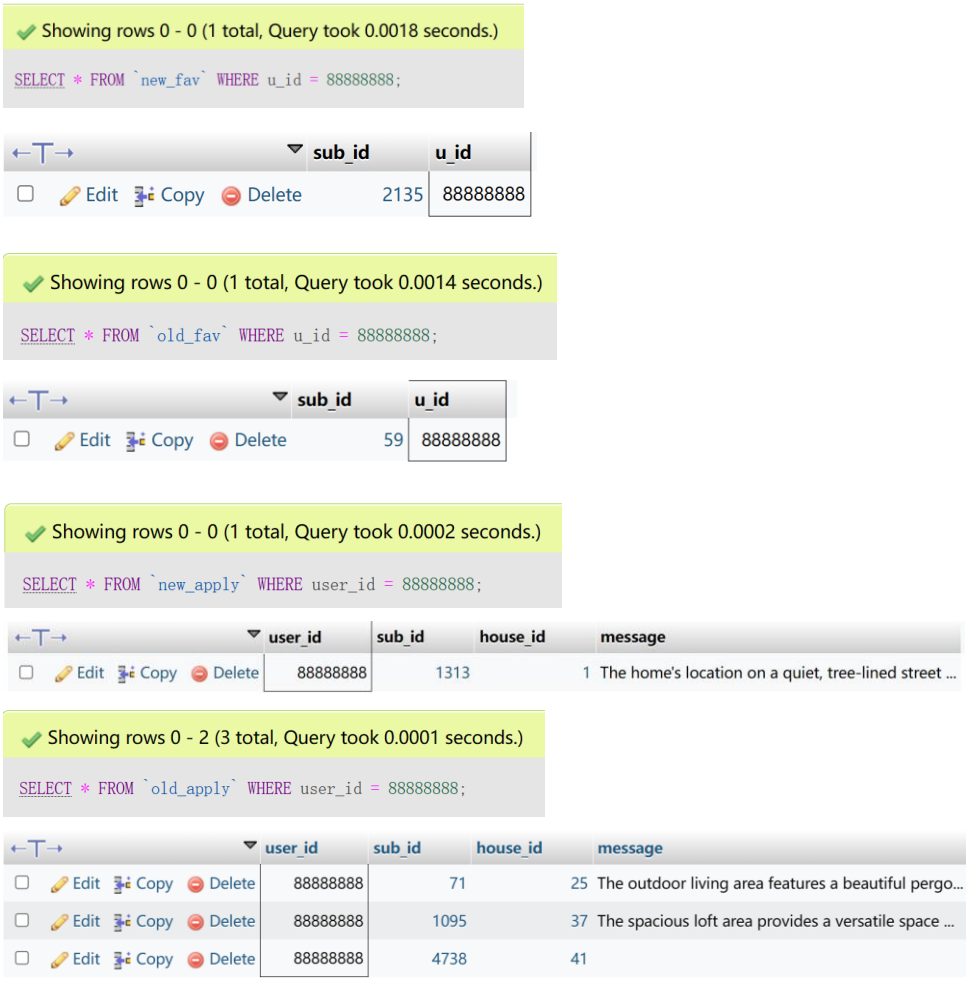
Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
-- 1 data SELECT * FROM `new_apply` WHERE user_id = 52195;
```

	user_id	sub_id	house_id	message
Edit Copy Delete	52195	1313	1	The home's location on a quiet, tree-lined street ...



After operations - Four tables:

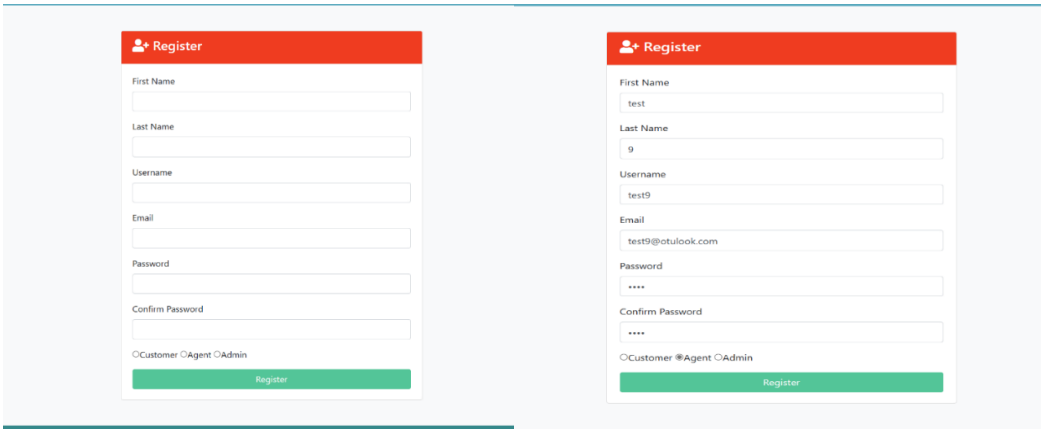


VII. Frontend Function

a) Registration and log in

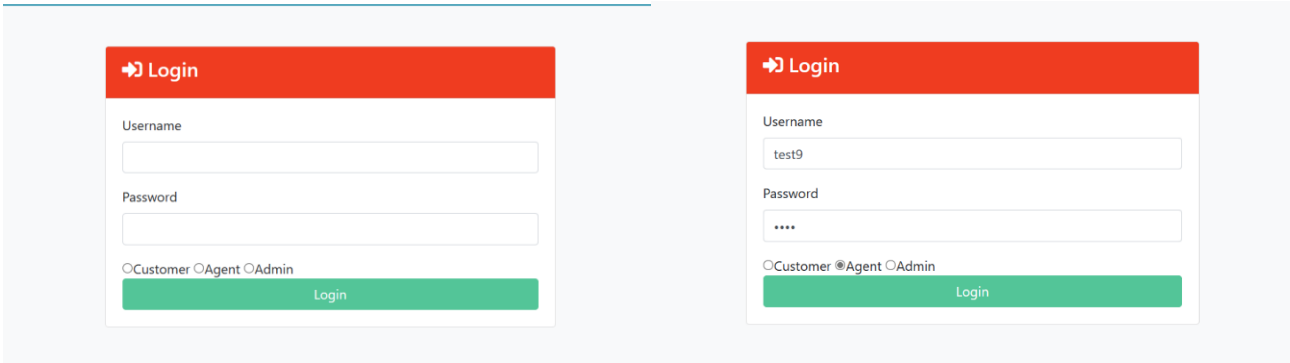
i. Registration

If a user wants to use the full functionality of our website, he/she must register at first. And when a user registers an account, there are three user identities available for the user to choose from, each of these three identities has different functions and permissions. The details will be shown to you later. And the following is our registration screen.



ii. Login

After the user completes registration, he can use the username and password of the account he just registered and select the corresponding identity to log in. Users with different identities will have different user dashboards after login. And the following is our login screen.



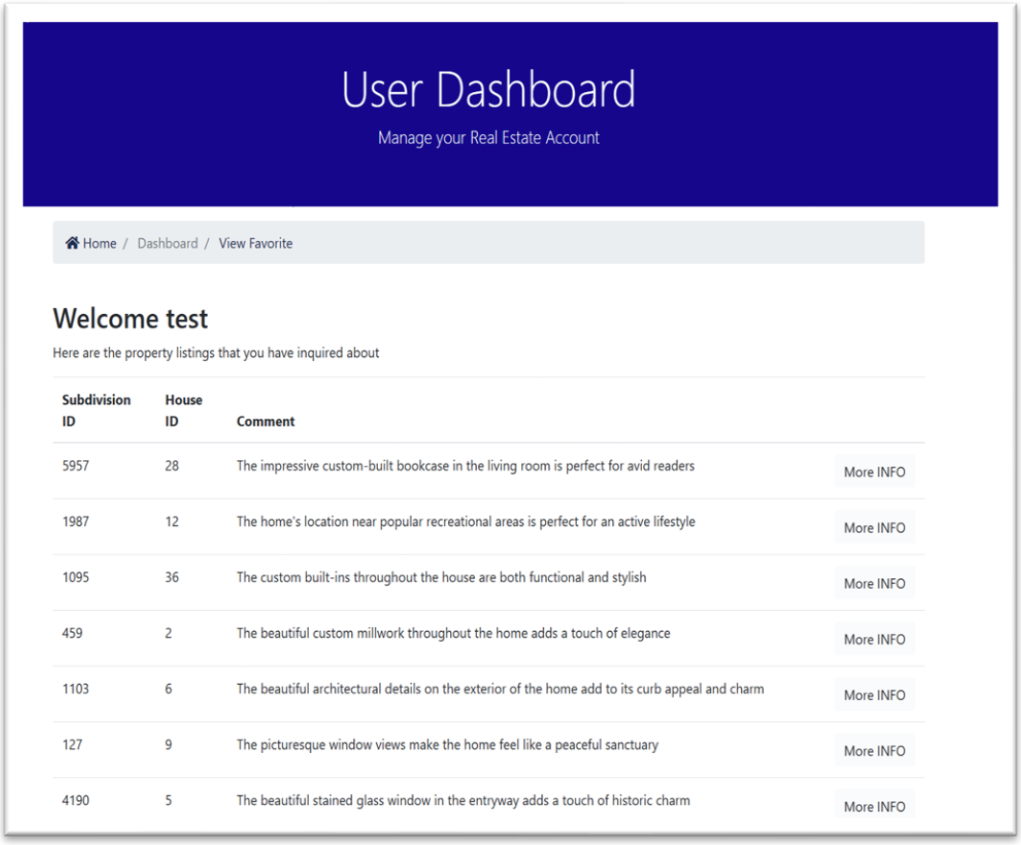
b) User Dashboard

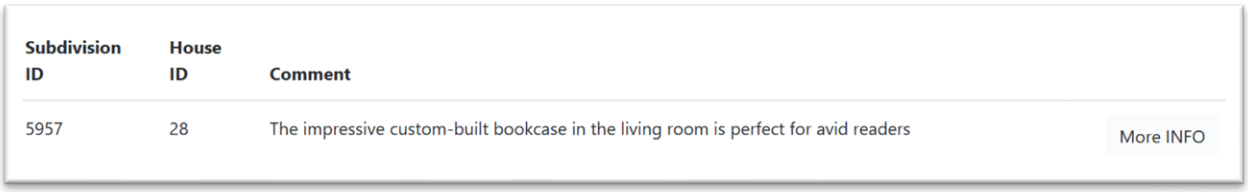
i. Customer Dashboard

If the user logs in as a customer, he/she will see the customer dashboard, in this user dashboard he/she can view the records of all the houses he has applied for (apply page), as well as all the subdivisions he/she has added to his/her favorites (favorite page).

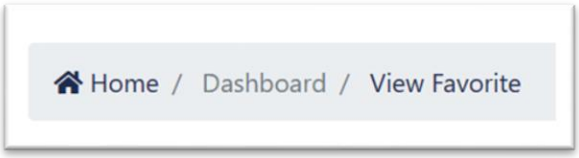
✧ Apply page

In the apply page, it lists the information including the subdivision ID, house ID, and the user's comments about the house according to the record of the house that the user has applied for. Also, at the end of each house listing, there is a "More INFO" button that users can click to go to the details of that house.



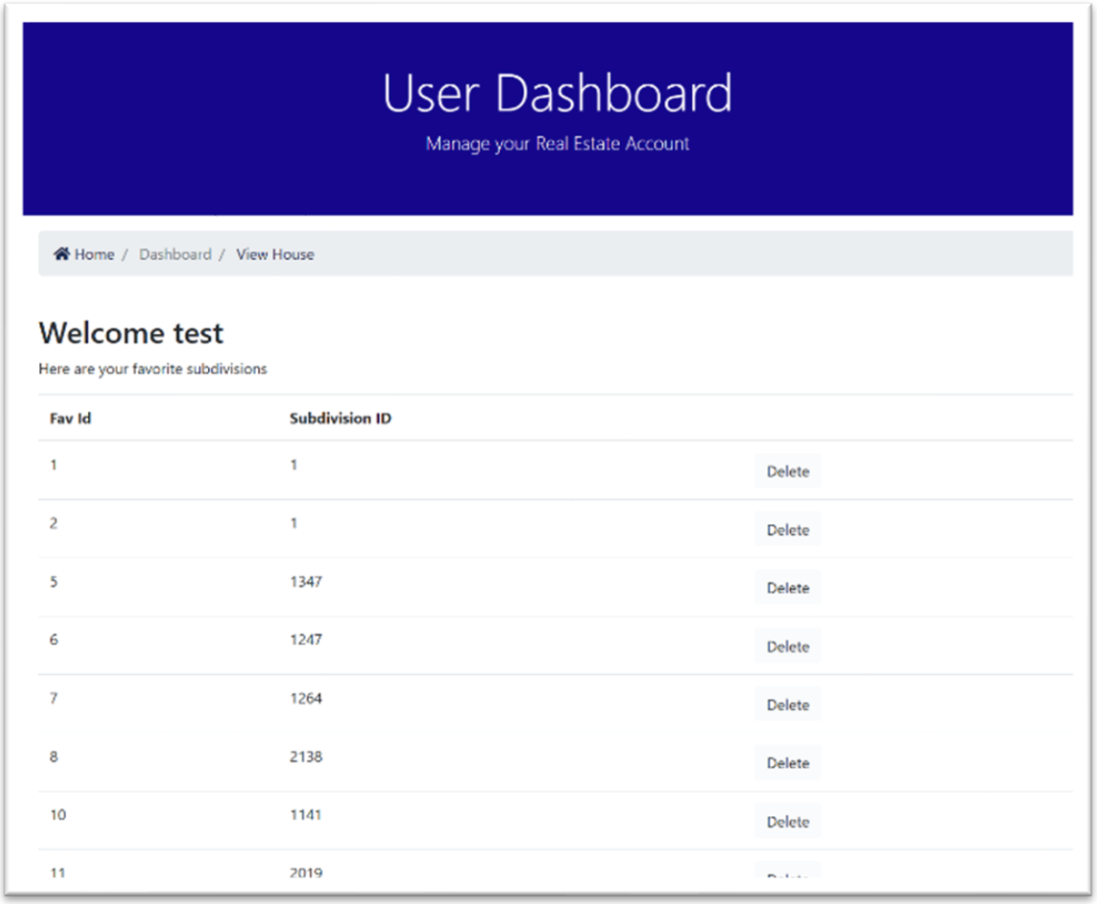


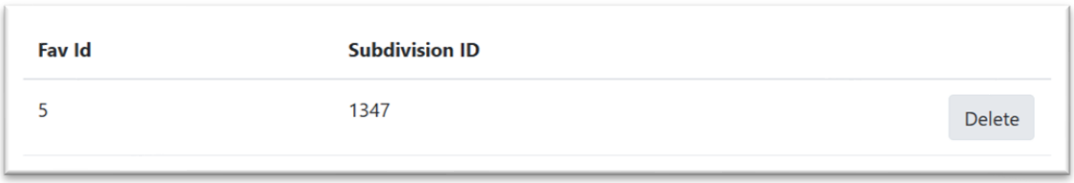
In addition, if users want to view their favorite subdivision, just click on the "View Favorite" button on the secondary navigation bar above to go to the favorites page.



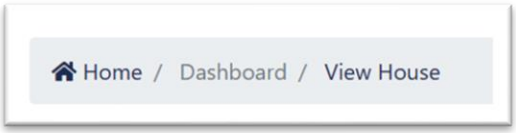
✧ Favorite page

In the favorite page, it lists the information including the favorite ID and subdivision ID, and the user's comments about the house according to the record of the subdivision that the user has favorited. Also, at the end of each favorite subdivision listing, there is a "Delete" button that users can click to delete the favorite.





In addition, if users want to view the houses that the user has applied for, just click on the "view House" button on the secondary navigation bar above to go to the apply page.

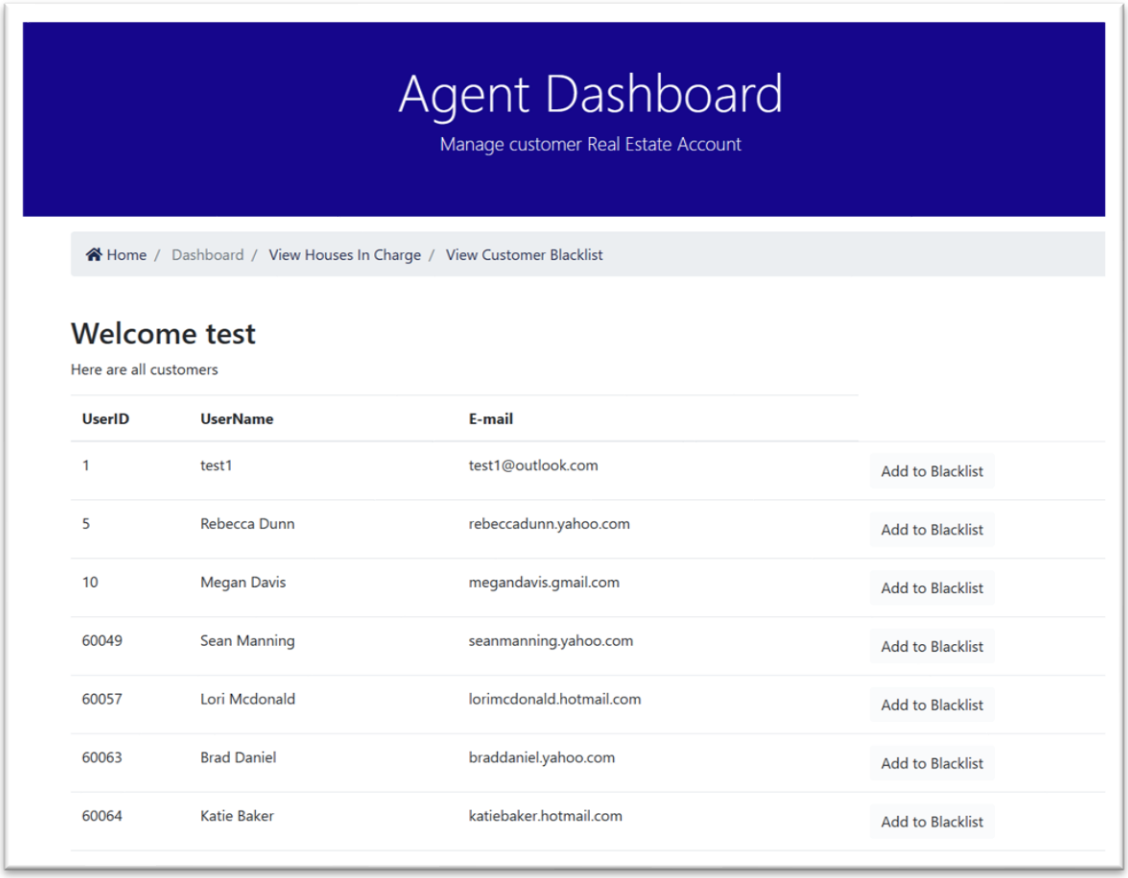


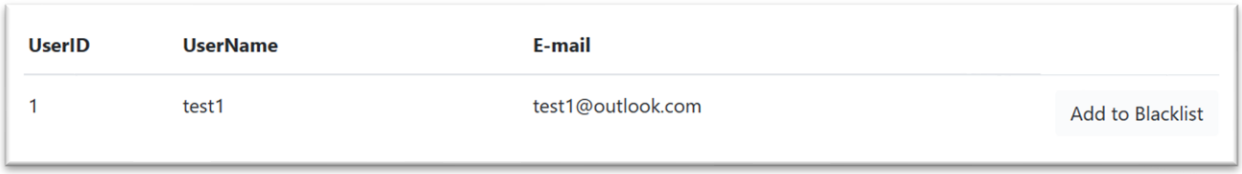
ii. Agent Dashboard

If the user logs in as an agent, he/she will see the agent dashboard, in this user dashboard he/she can view the information of all registered non-blacklisted customers (All Customers page), information of all houses that are managed by the user (House in Charge page), and information of all blacklisted customers (Customer Blacklist page).

✧ All Customers page

In the all-customers page, it lists the information including the user ID, user name, and the user's e-mail. Also, at the end of each customer listing, there is a "Add to Blacklist" button that the user can click to add the customer to the blacklist. Attention, once a customer is added to the blacklist, the customer's account will not be able to log in, which means that the user loses the right to use our website.

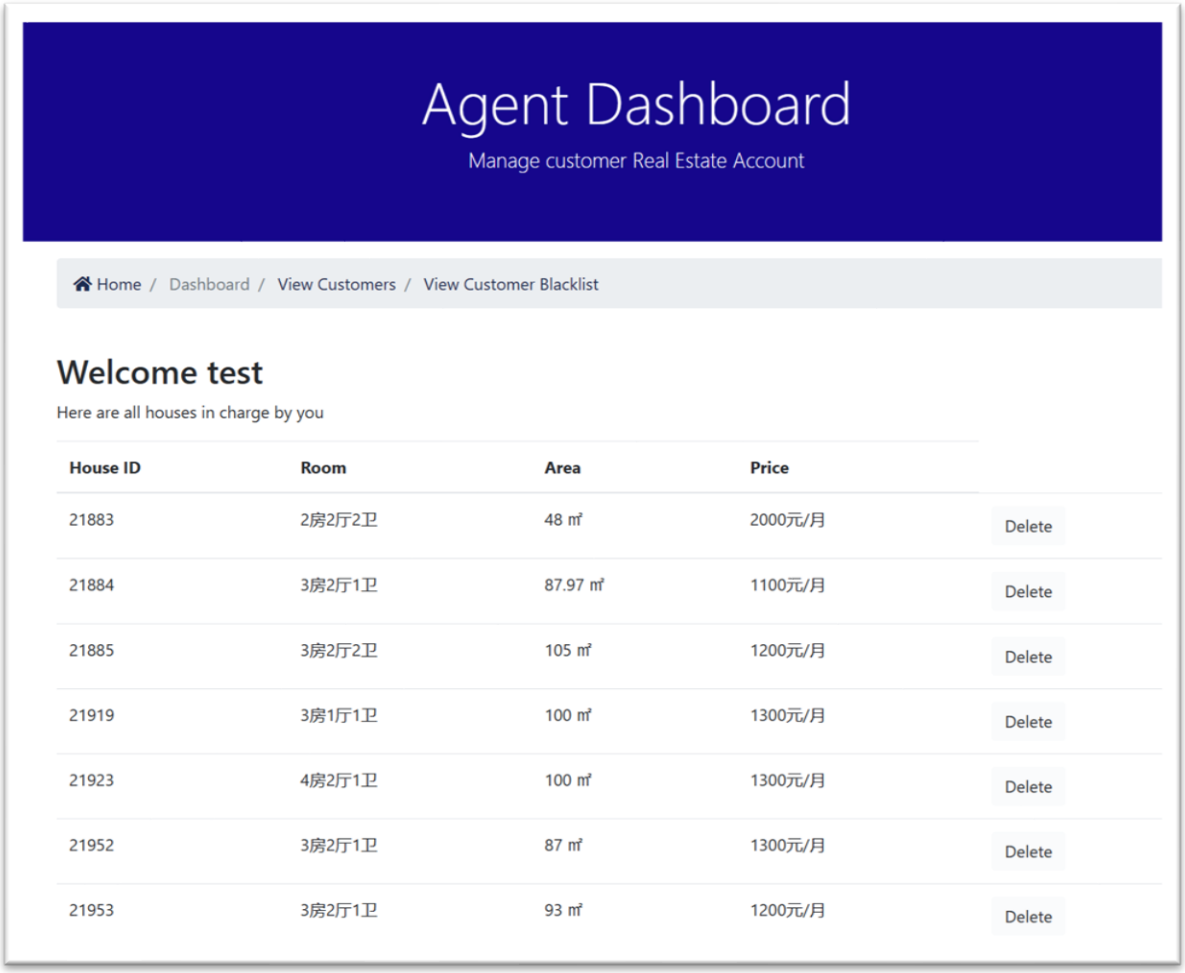


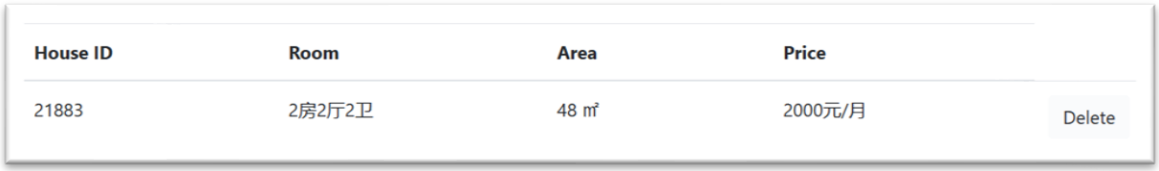


In addition, if users want to view the information of all houses that are manage by the themselves or view the information of all blacklisted customers, just click on the "View Houses In Charge" button or the "View Customer Blacklist" button on the secondary navigation bar above to go to the other page.

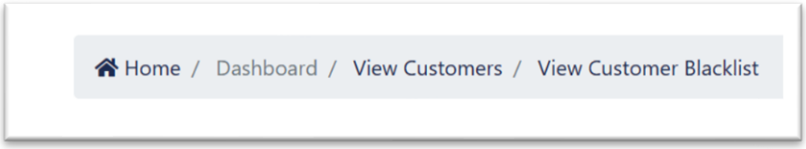
✧ House in Charge page

In the house in charge page, it lists the information including the house ID, room number, house area, and house price. Also, at the end of each house listing, there is a "Delete" button that the user can click to delete the house information. Attention, once a property is deleted, it will be taken down from the list of available houses listed in our houses listing screen.



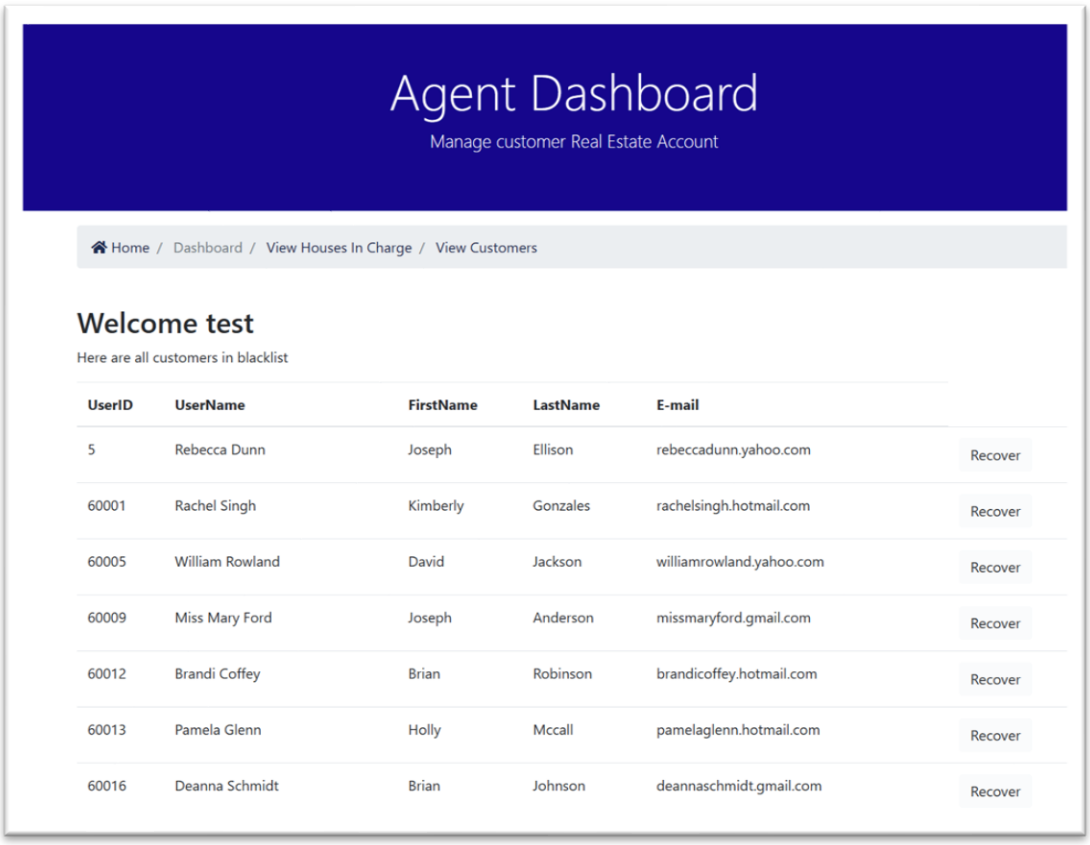


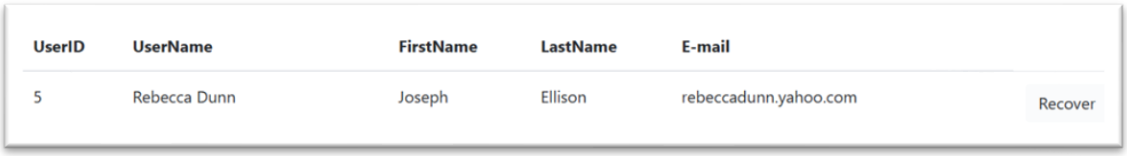
In addition, if users want to view the information of all registered non-blacklisted customers or view the information of all blacklisted customers, just click on the "View Customers" button or the "View Customer Blacklist" button on the secondary navigation bar above to go to the other page.



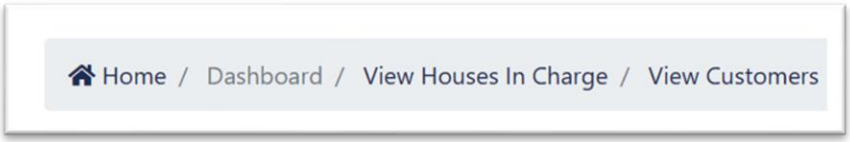
✧ **Customer Blacklist page**

In the customer blacklist page, it lists the information including the user ID, user name, user’s first name, user’s last name, and user’s e-mail. Also, at the end of each blacklist customer listing, there is a "Recover" button that the user can click to recover the customer from the blacklist. Attention, once a customer is recovered from the blacklist, the customer's account will be able to log in, which means that the user recovers the right to use our website.





In addition, if users want to view the information of all registered non-blacklisted customers or view the information of all houses that are managed by themselves, just click on the "View Customers" button or the "View Houses In Charge" button on the secondary navigation bar above to go to the other page.



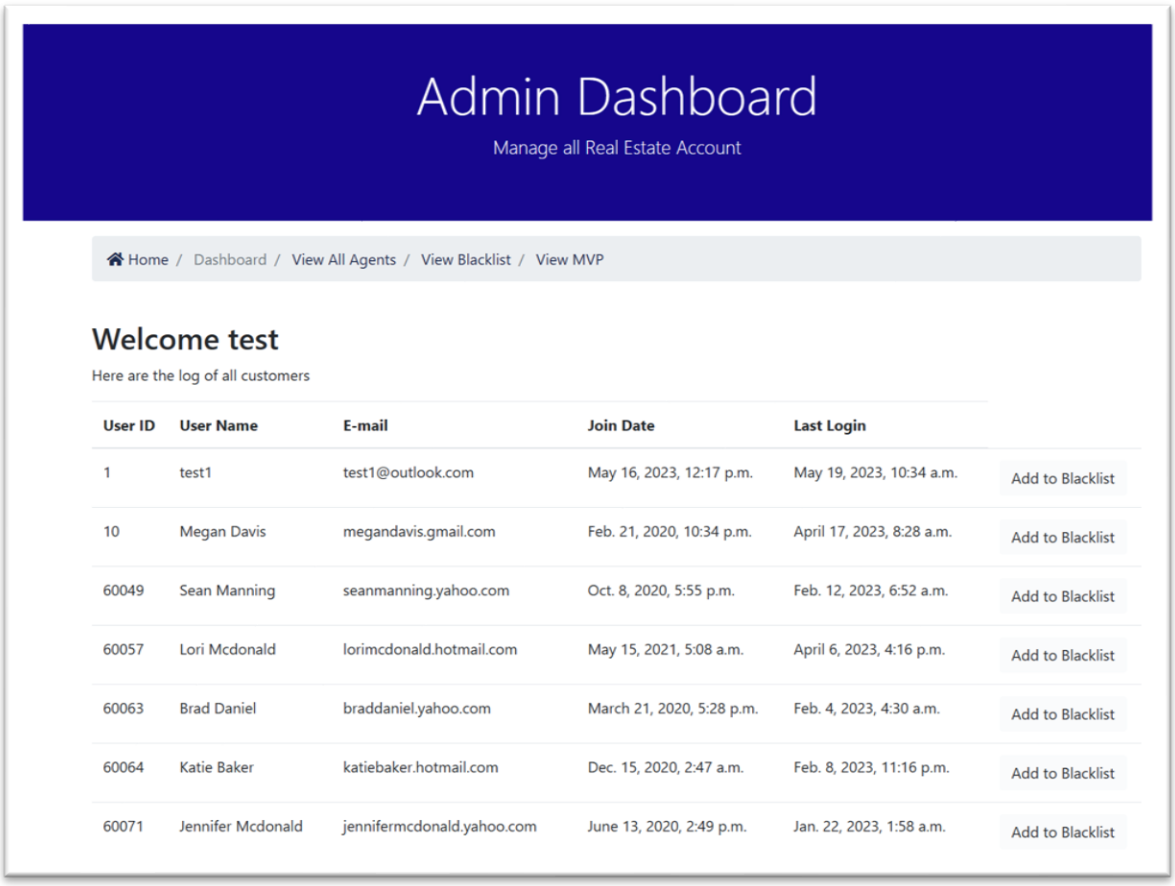
iii. Admin Dashboard

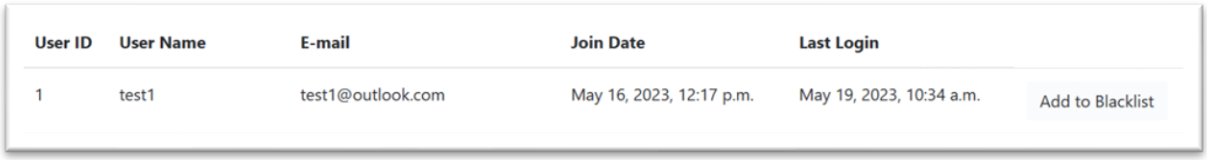
If the user logs in as an admin, he/she will see the admin dashboard, in this user dashboard he/she can view the information of all registered non-blacklisted customers (All Customers page), the information of all registered non-blacklisted agents (All Agents page), the information of all blacklisted users (All-Blacklist page), and the information of all agents who were awarded MVP (MVP page).

✧ All Customers page

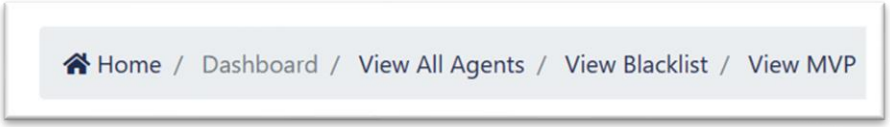
In the all-customers page of the admin, it lists the more detail information of customers including the user ID, user name, user's e-mail, user’s join date and the user’s last login date.

Also, at the end of each customer listing, there is a "Add to Blacklist" button that the user can click to add the customer to the blacklist. Attention, once a customer is added to the blacklist, the customer's account will not be able to log in, which means that the user loses the right to use our website.





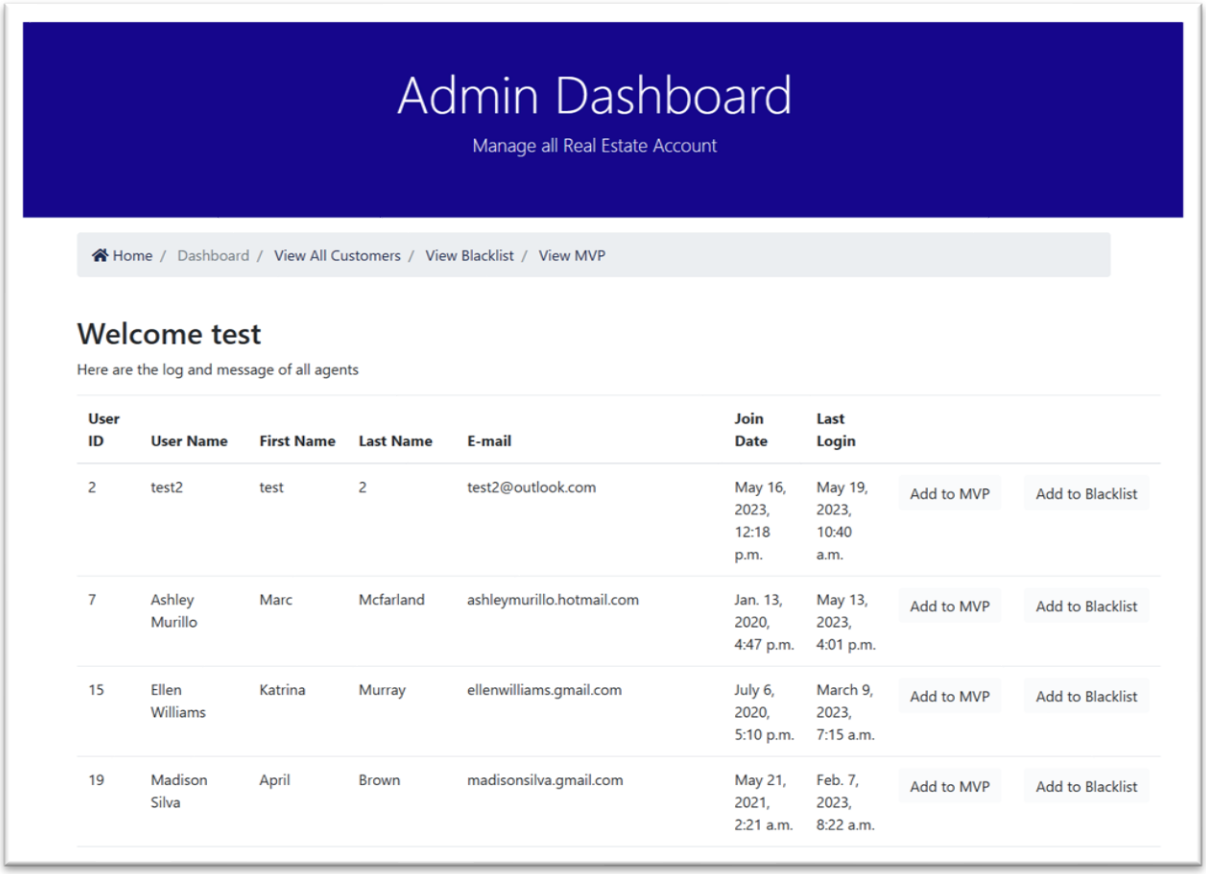
In addition, if users want to view the information of all registered non-blacklisted agents or the information of all blacklisted users or the information of all agents who were awarded MVP, just click on the "View All Agents" button or the "View Blacklist" button or the "View MVP" button on the secondary navigation bar above to go to the other page.

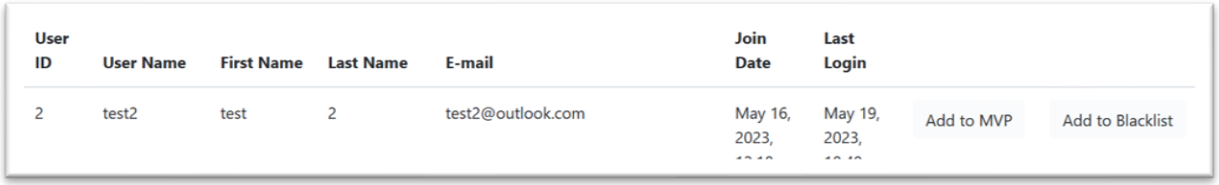


✧ All Agents page

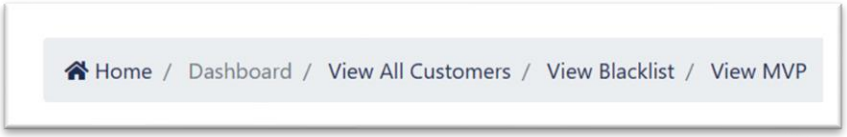
In the all-agents page of the admin, it lists the detailed information of agents including the user ID, user name, user's first name, user's last name, user's e-mail, user's join date, and the user's last login date. Also, at the end of each customer listing, there are two button which are "Add to MVP" button and "Add to Blacklist" button. The user can click the "Add to MVP" button to add the agent to be the MVP, and can click the "Add to Blacklist" button to add the

agent to the blacklist. Attention, after an agent be added to be the MVP, he/she's information will be listed in the MVP page. Also, once an agent is added to the blacklist, the agent's account will not be able to log in, which means that the user loses the right to use our website.





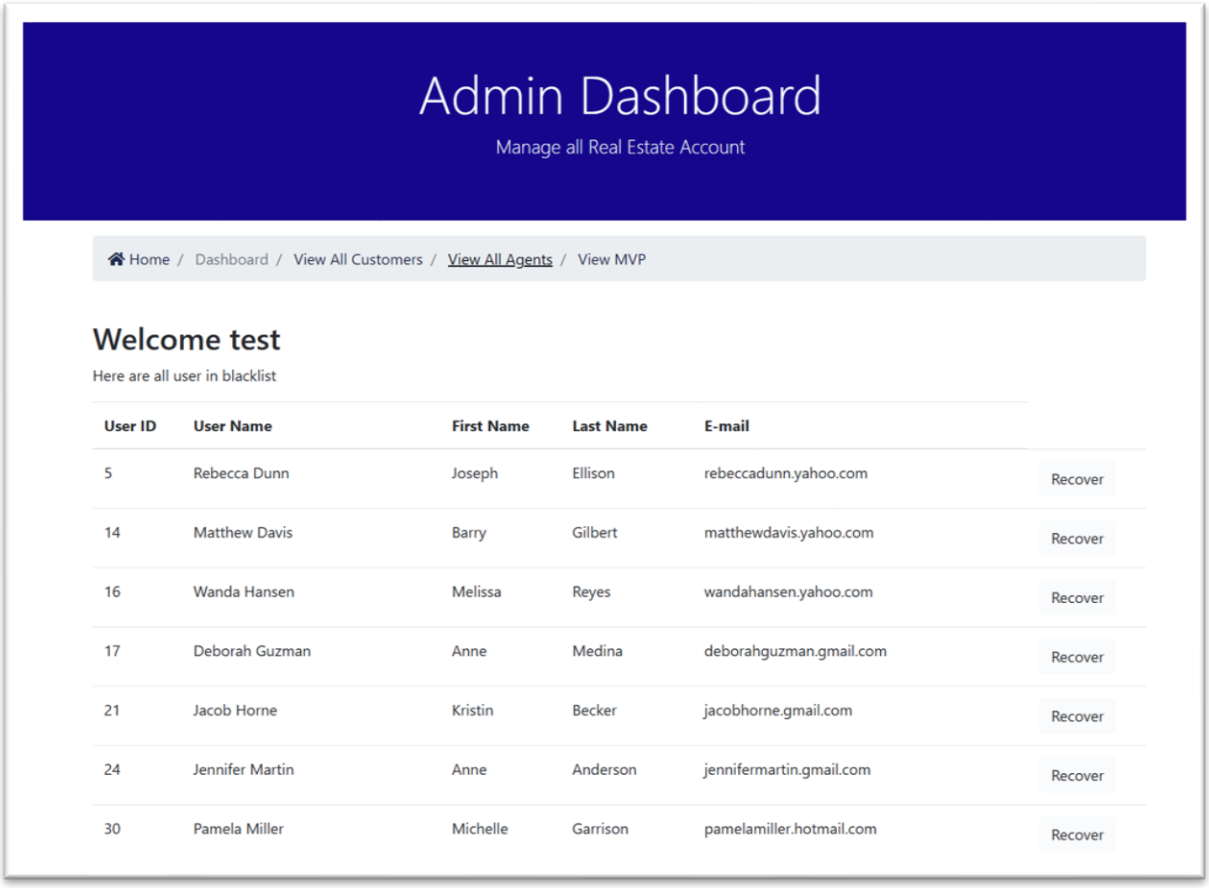
In addition, if users want to view the information of all registered non-blacklisted customers or the information of all blacklisted users or the information of all agents who were awarded MVP, just click on the "View All Customers" button or the "View Blacklist" button or the "View MVP" button on the secondary navigation bar above to go to the other page.

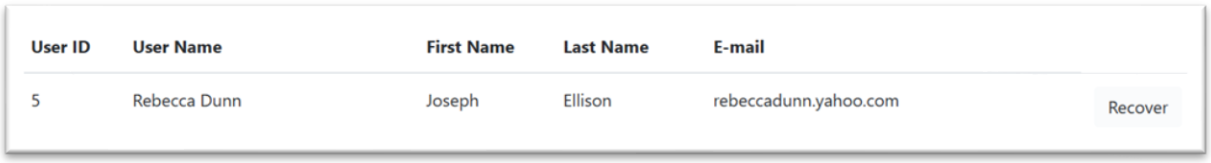


✧ All-Blacklist page

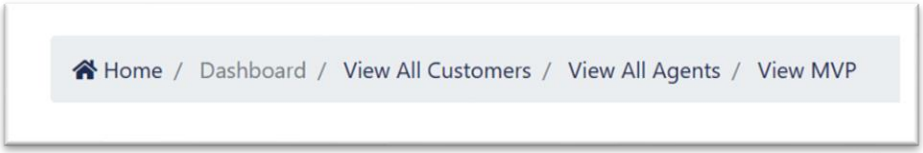
In the all-blacklist page, it lists the information including the user ID, user name, user's first name, user's last name, and user's e-mail. Different from the blacklist in the agent dashboard, the all-blacklist page in the admin dashboard including the customer blacklist and the agent blacklist. Both of the customer and the agent who is in the blacklist will be listed in this page. Also, at the end of each blacklist customer listing, there is a "Recover" button that

the user can click to recover the user from the blacklist. Attention, once a user is recovered from the blacklist, the user's account will be able to log in, which means that the user recovers the right to use our website.





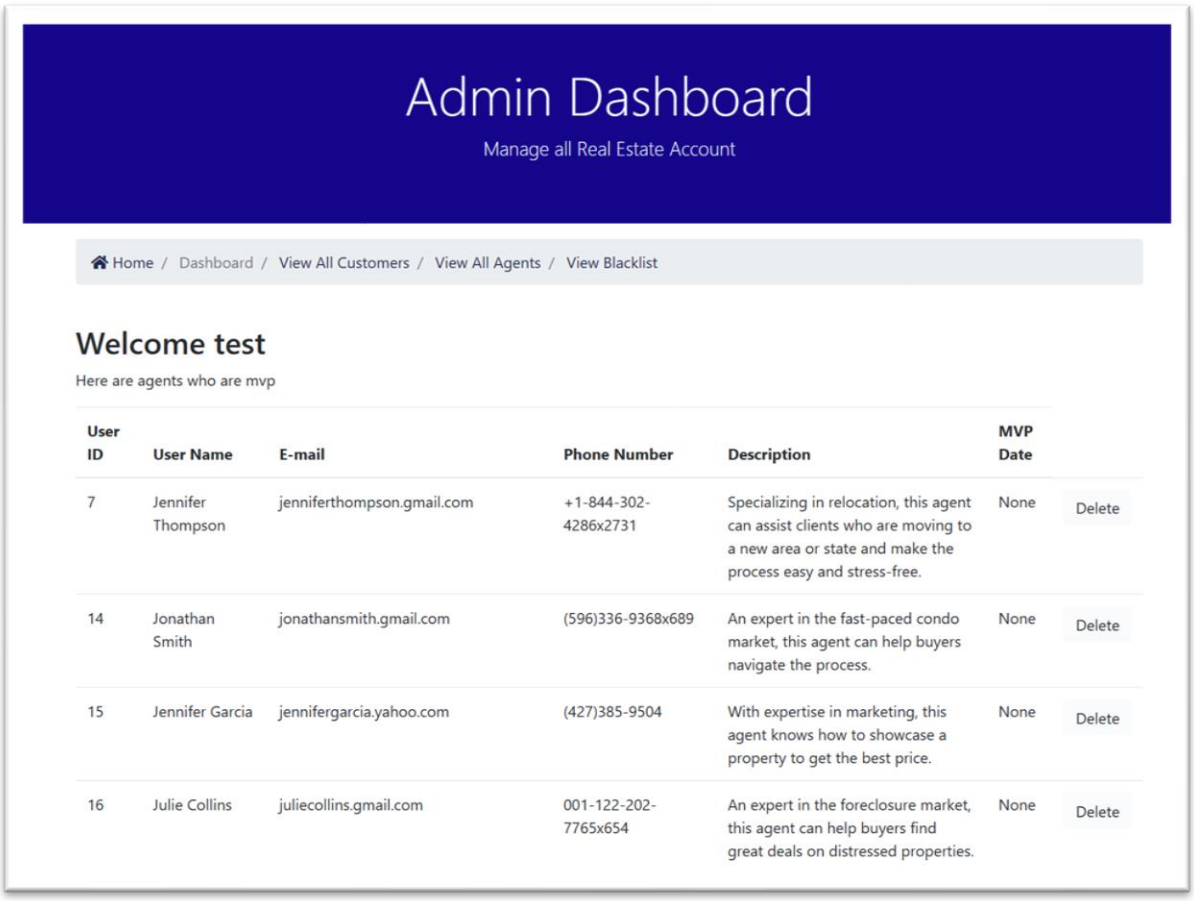
In addition, if users want to view the information of all registered non-blacklisted customers or the information of all registered non-blacklisted agents or the information of all agents who were awarded MVP, just click on the "View All Customers" button or the "View All Agents" button or the "View MVP" button on the secondary navigation bar above to go to the other page.



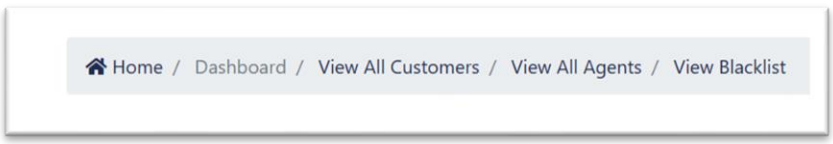
✧ **MVP page**

In the MVP page of the admin, it lists the detailed information of agents who were awarded MVP including the user ID, user name, user's e-mail, user's phone number, the description to this user, and the date that the agent was awarded the MVP. Also, at the end of each MVP

listing, there is a "Delete" button that the user can click to delete the MVP. Attention, once an MVP is deleted, it means that the agent will lose the honor forever until the next time the MVP is awarded.



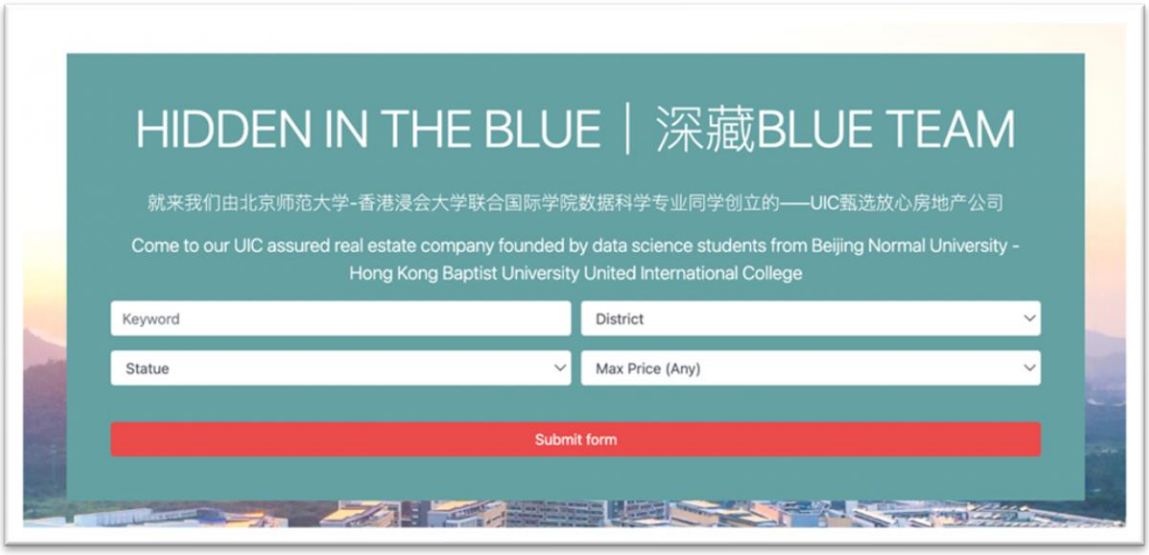
In addition, if users want to view the information of all registered non-blacklisted customers or the information of all registered non-blacklisted agents or the information of all blacklisted users, just click on the "View All Customers" button or the "View All Agents" button or the "View Blacklist" button on the secondary navigation bar above to go to the other page.

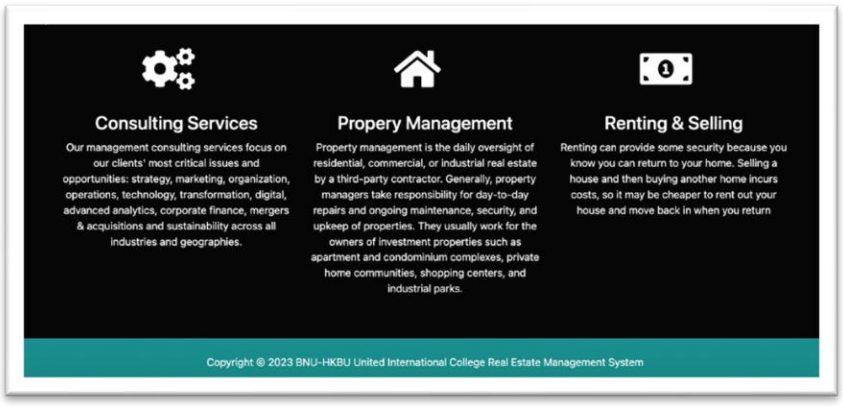


c) Index Page

The first thing you will see on the homepage is a search box which allows you to filter by keyword, new or second hand, district of the house and the highest price, next down is a selection of our sponsor properties, in this screen we can view the details of the sponsor properties and see the properties for sale

in the sponsor properties, further down are some descriptions of the real estate. In the back, a function called `index` that handles an HTTP request to display a web page. It retrieves data from a database table named `listings_sub_div` using a raw SQL query and creates a list of objects of type `sub_div` from the query results. The function then creates a dictionary named `context` containing the `listings` object and several other variables, which are used to populate UI elements on the web page. Finally, the function returns an HTTP response object generated by calling the `render()` function with the `request` object, the name of the HTML template file to use, and the `context` dictionary as arguments.

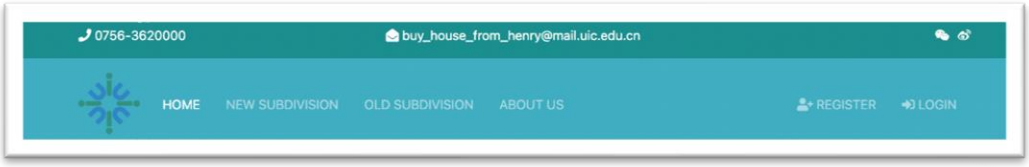




d) Navigation Bar

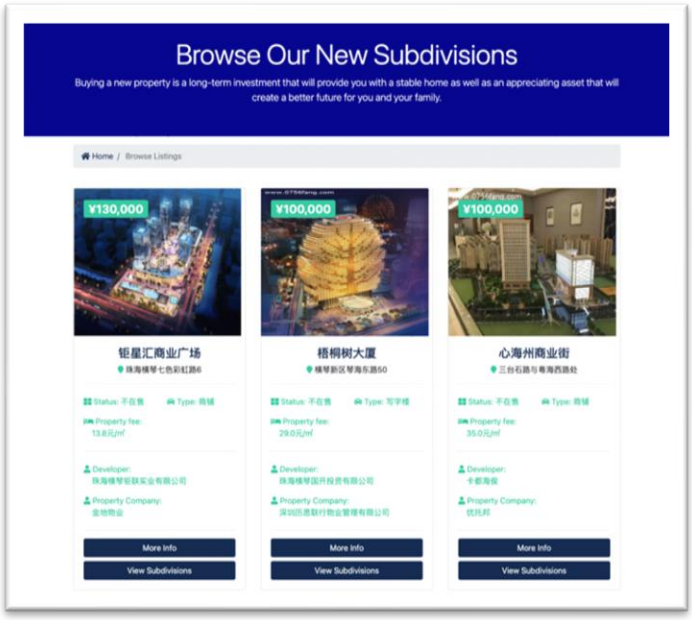
From left to right there is the logo, the Home Page, the New Subdivision screen, the Old

Subdivision screen, the About us page, the Personal page, the Registration, Login and Logout screens. In the back, we use the URL redirection.



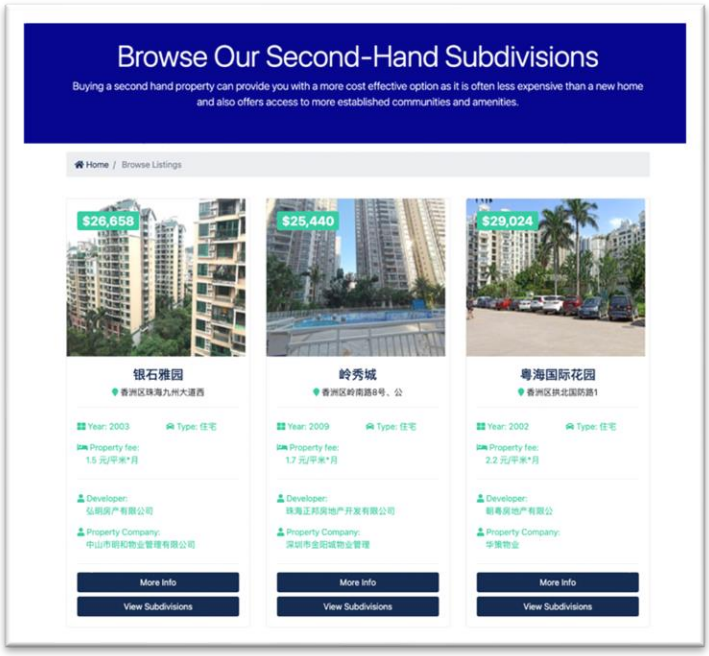
e) New Subdivision Page

In the New Subdivisions screen we can display all the new subdivisions that exist in the database, 6 subdivisions can be displayed on one page and there is a page turn button at the bottom of the screen. Each plot can display the name of the plot, the address of the plot, whether it is for sale or not, whether it is a shop or a commercial property, the cost of the property, the developer, the property, etc. Below this there are two buttons to display more information about the subdivisions and to view all properties in the subdivisions. In the back, a function called `listing_div` that retrieves a specific `sub_div` object from the database using its ID, adds it to a dictionary named `context`, and passes it to the `render()` function to render an HTML template with the context data and return it as an HTTP response.



f) Old Subdivision Page

Same as new subdivision page, In the Old Subdivisions screen we can display all the old subdivisions that exist in the database, 6 subdivisions can be displayed on one page and there is a page turn button at the bottom of the screen. Each plot can display the name of the plot, the address of the plot, whether it is for sale or not, whether it is a shop or a commercial property, the cost of the property, the developer, the property, etc. Below this there are two buttons to display more information about the subdivisions and to view all properties in the subdivisions. In the back, a function called `listing_div` that retrieves a specific `sub_div` object from the database using its ID, adds it to a dictionary named `context`, and passes it to the `render()` function to render an HTML template with the context data and return it as an HTTP response.

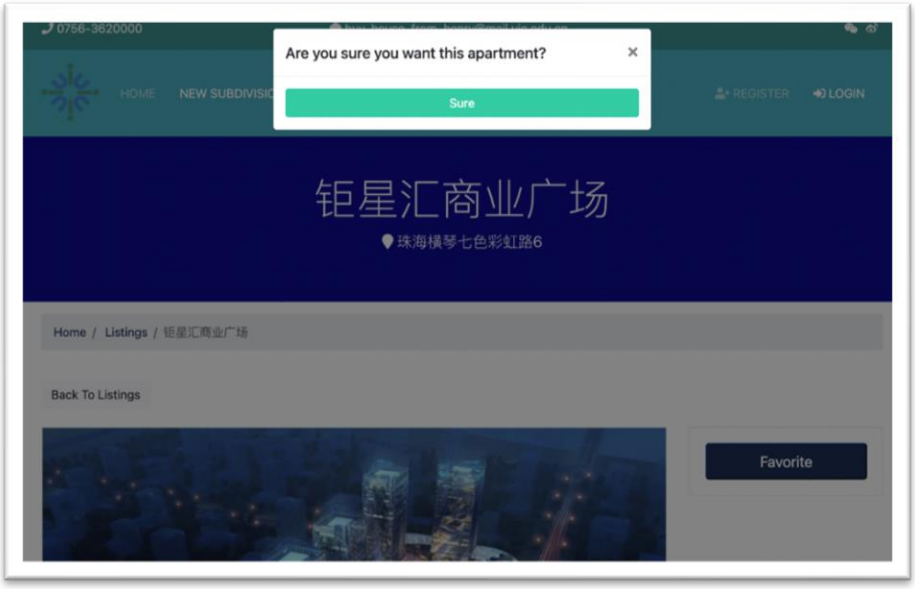


g) More Information Page of New/Old Subdivision

We show pictures of the corresponding old and new subdivisions, as well as various details in our database, including prices, property rates, for sale/not for sale, property companies, development companies, transport, subdivisions environment, subdivisions entertainment, and subdivisions businesses. In the back, a function called `listing_div` that retrieves a specific `sub_div` object from the database using its ID, adds it to a dictionary named `context`, and passes it to the `render()` function to render an HTML template with the context data and return it as an HTTP response.

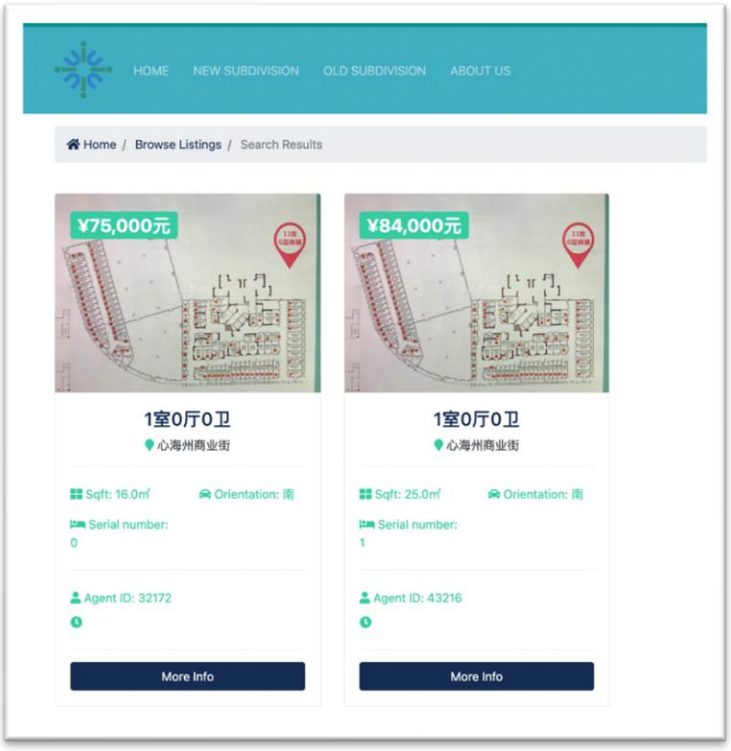
h) Favorite

We can bookmark/favorite any subdivision in the more information page for old and new subdivisions. In the back, a function called `fav` handles a POST request to collect a listing. It checks if the user has already collected the listing and displays an error message if so, otherwise, it creates a new `Fav` object and saves it to the database. The user is then redirected to the listing page. The function doesn't handle any GET requests.



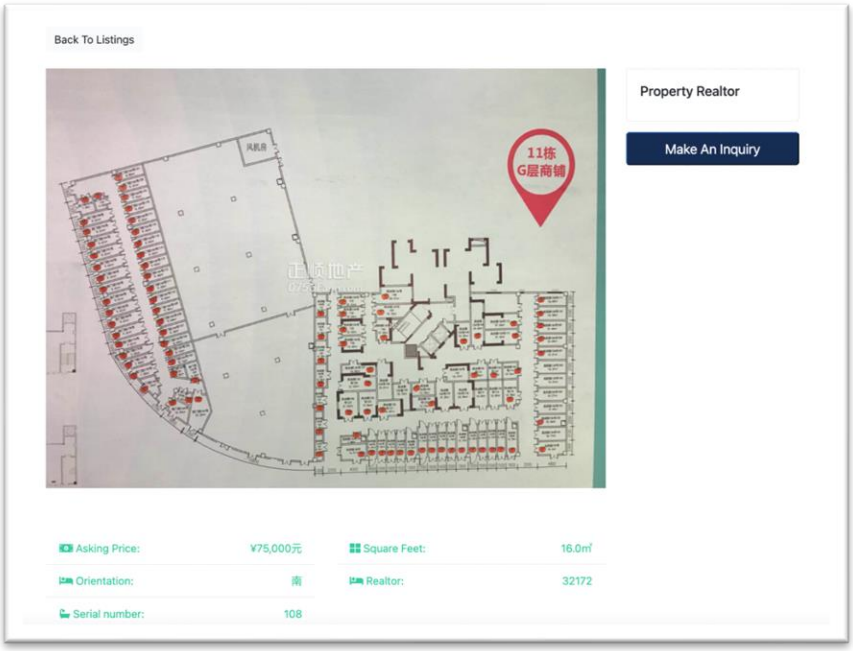
i) View New/Old Subdivision Page

This page shows all the properties under this subdivision, as well as some details of the properties, including price, area, orientation, etc. In the back, a function called `search_div_old` retrieves the data that matches a specified `listing_div_id` from the database using a raw SQL query. It then creates a list of `Listing_old` objects from the query results and adds them to a dictionary named `context`. Finally, the function passes the `context` dictionary to the `render()` function to render an HTML template and return it as an HTTP response.



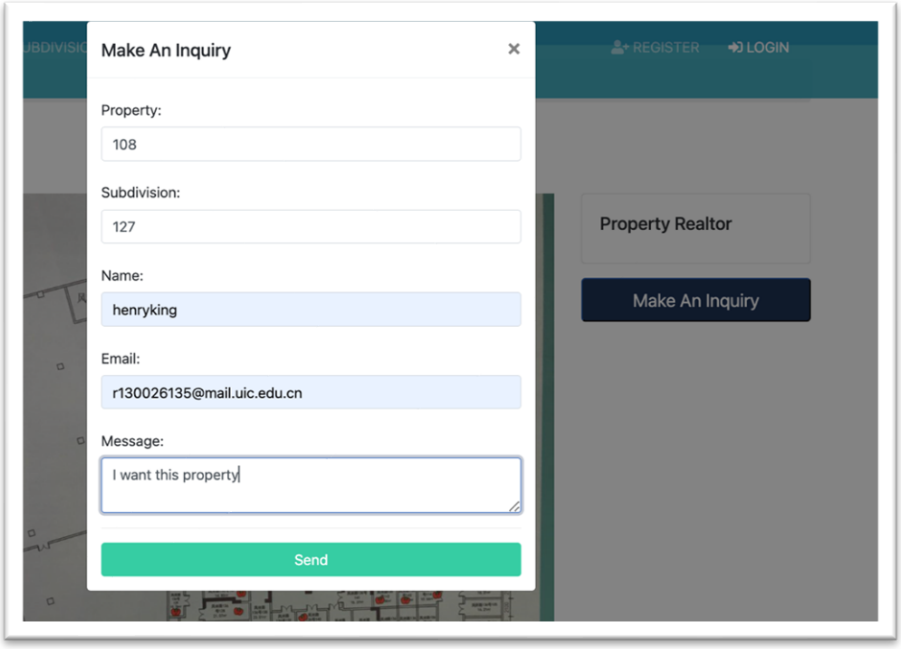
j) More Information Page of New/Old Properties

This page shows the details of the property, including pictures, price, size, orientation, etc. We can also apply for the property in this page. In the back, a function retrieves a specific `sub_div` object from the database using its ID, adds it to a dictionary named `context`, and passes it to the `render()` function to render an HTML template with the context data and return it as an HTTP response.



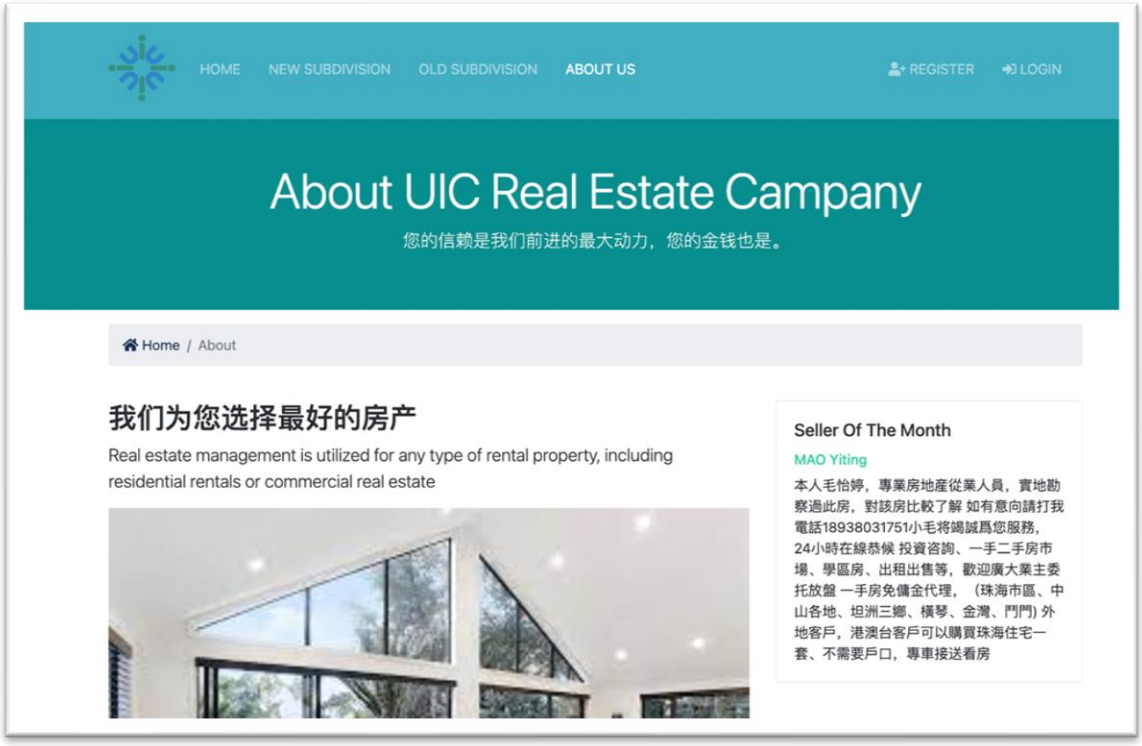
k) Make An Inquiry Page

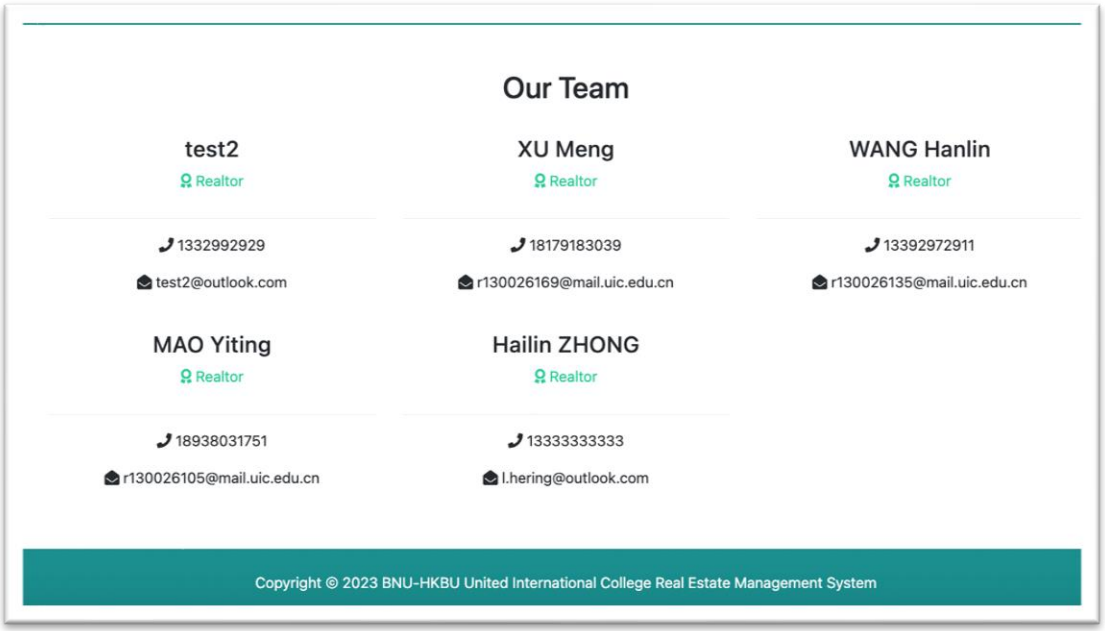
In this page, the property number and block number are automatically filled in according to the property you are in, and if you are logged in, your name and email address are also automatically filled in, and you can add some additional information. In the back, a function called `contact` that handles a POST request to submit a contact inquiry about a listing. It checks if the user has already made an inquiry, and if not, creates a new `Contact ` object and saves it to the database. The user is then redirected to the listing page. The function doesn't handle any GET requests.



l) About Us Page

On this page, we show the current monthly top sellers, as well as all our agents. In the back, a function called `about` handles a request to display an "About Us" page. The function retrieves data from the `realtors_realtor` table using raw SQL queries and the `connection.cursor()` method. It retrieves all the realtors and MVPs using separate queries. The query results are then converted into a list of `Realtor` objects using a list comprehension. The `Realtor` objects are created by unpacking the tuple using the `*` operator and passing the resulting values as arguments to the `Realtor` constructor. The function then creates a dictionary named `context` containing the `realtors` and `mvp_realtors` objects. Finally, the function returns an HTTP response object generated by calling the `render()` function with the `request` object, the name of the HTML template file to use (`pages/about.html`), and the `context` dictionary as arguments. The `render()` function takes care of rendering the template with the context data and returning the resulting HTML as the HTTP response.





Contribution

Name	Contribution
Mao Yiting	● Crawler & Data Generation ● ER Diagram ● Database (Constraints & Trigger, Frontend SQL Query) ● Frontend & backend (Pageturner) ● Report (I-IV, V(except part c), VI(part c)) & Integrate Document ● PPT (part 1-3)
Xu Meng	● Data Cleaning ● Database (Construction) ● Report (V(part c), VI(except part c)) ● PPT (part 4 & 5)
Zhong Hailin	● Frontend & Backend (Register & Login Function) ● Frontend & Backend (Cutomer & Agent & Admin Dashboad) ● Frontend & Backend (View Apply/Favorite Function) ● Frontend & Backend (View House/Customer/Blacklist Function) ● Frontend & Backend (View Customer/Agent/Blacklist/MVP Function) ● Frontend & Backend (More Info Function) ● Frontend & Backend (Delete Favorite Function) ● Frontend & Backend (Delete House Function) ● Frontend & Backend (Add/Recover to Blacklist Function) ● Frontend & Backend (Add/Delete to MVP Function) ● Report (VII (part a & b)) ● PPT (part 6)
Wang Hanlin	● Frontend-backend (Index Page with Search Engine) ● Frontend-backend (New/Second-Handed Subdivisions Page) ● Frontend-backend (New/Second-Handed Properties Page) ● Frontend-backend (Favorite Any Properties) ● Frontend-backend (Apply Any Properties) ● Frontend-backend (About us Page) ● Report (VII (except part a & b)