# Database final report

Members: Guofeng Tang, Jiacheng Xue

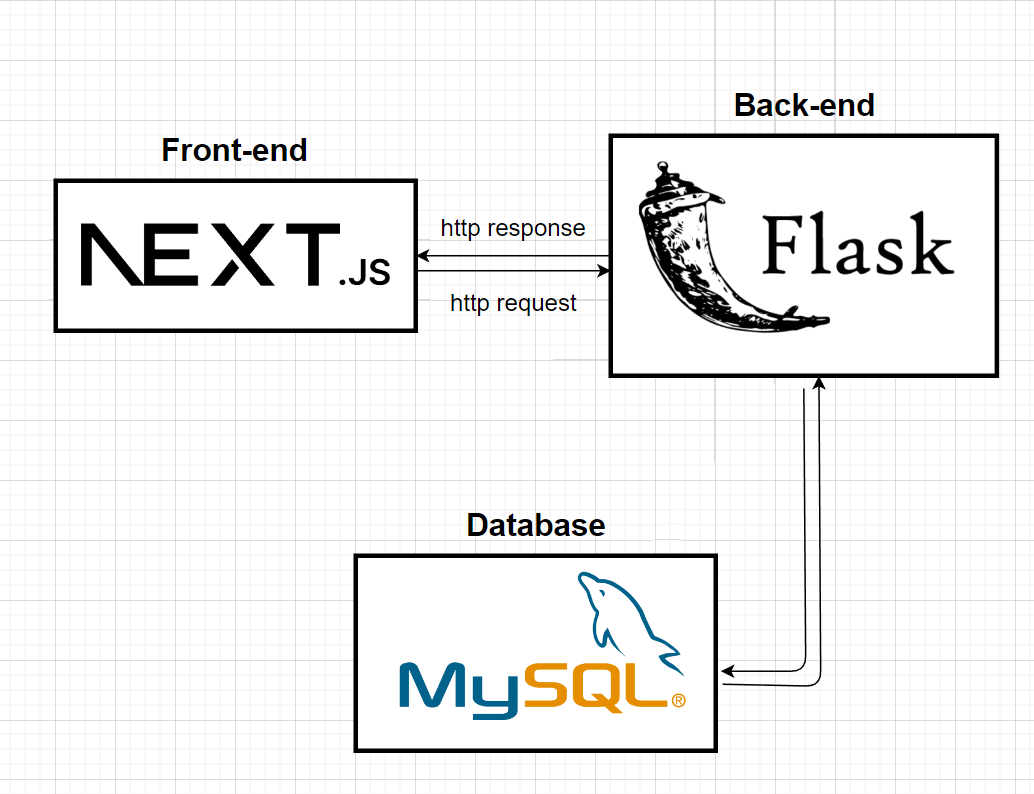
1. **Work task allocation**

Jiacheng Xue: All front-end work, database design, and functional testing.

Guofeng Tang: All backend work, database design, documentation writing.

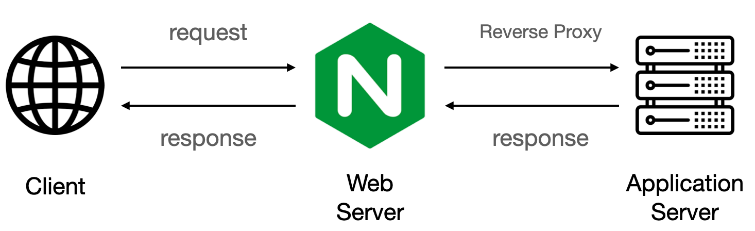
1. **System Implementation**

Front-end and back-end connection:



Img1 Front-end and back-end connection diagram

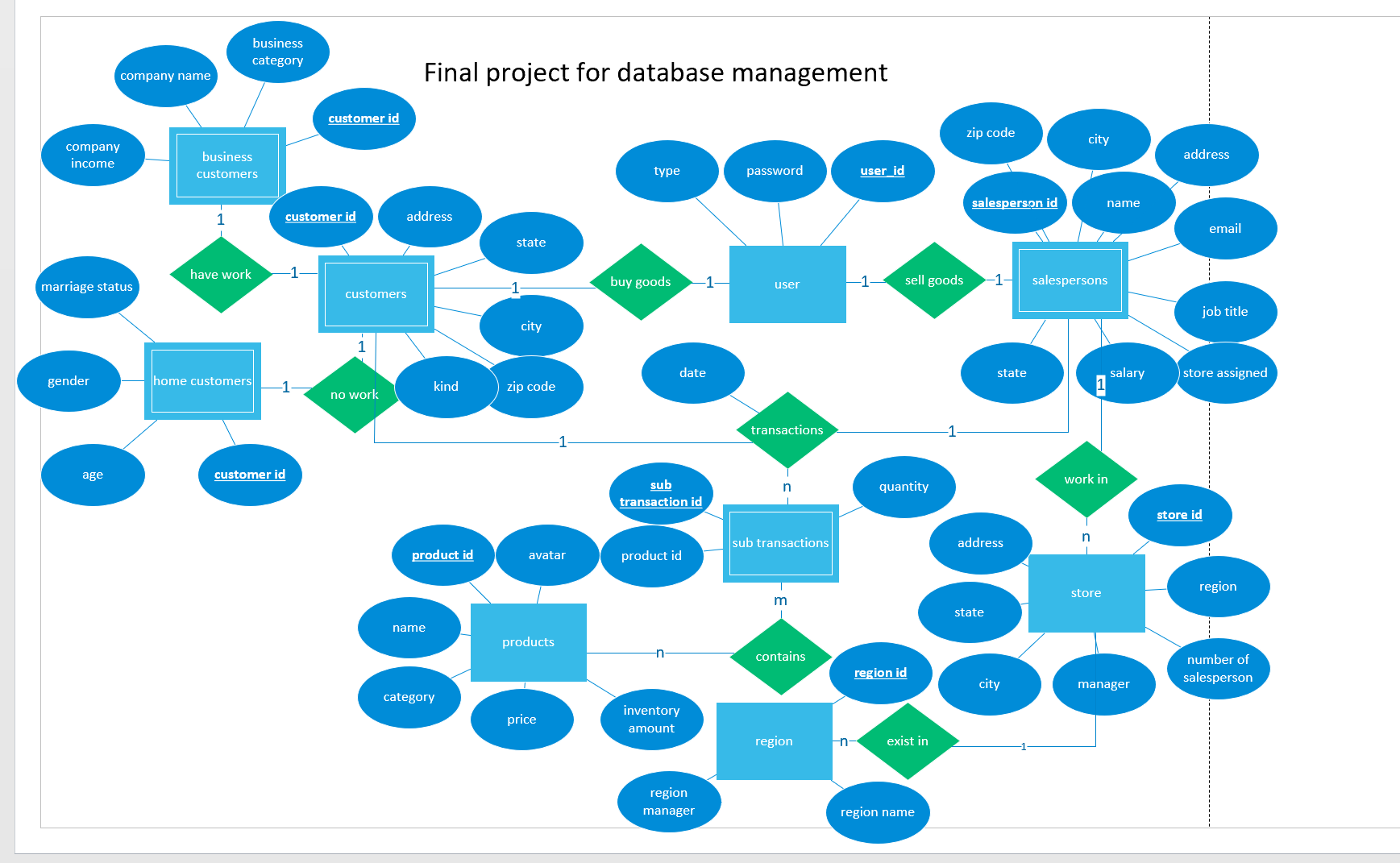
We use next.js as frontend framework and flask as backend. We use mysql as database and connect it with backend. Front-end request data from backend through http protocol.



Img2 Introduction to connection methods

More specifically, we have deployed our app behind nginx, which make user able to access both frontend and backend under same domain.

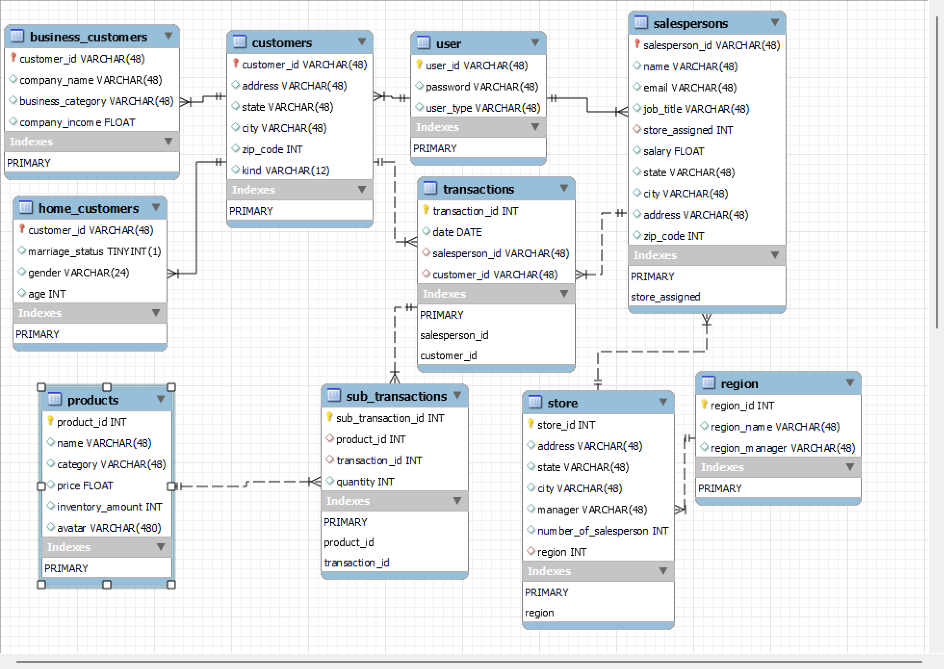
1. **E-R table**



Img3 E-R table

The entity sets are user, customers, business customers, home customers, salespersons, store, region, sub transactions and products. User contains user id, password and type. Type can only be input as “customer” or “salesperson”. Customers contains address information and the kind. Kind can only be input as “business” and “home”, so that we can judge if the customer is business customers or home customers. Salespersons include the address information, name, email, job, salary and store. The store information is stored in the store table, with address information, manager, number of salesperson and region. The region information is stored in region table, containing region manager, region name and region id. And we have sub transactions and products for us database to create the transactions.

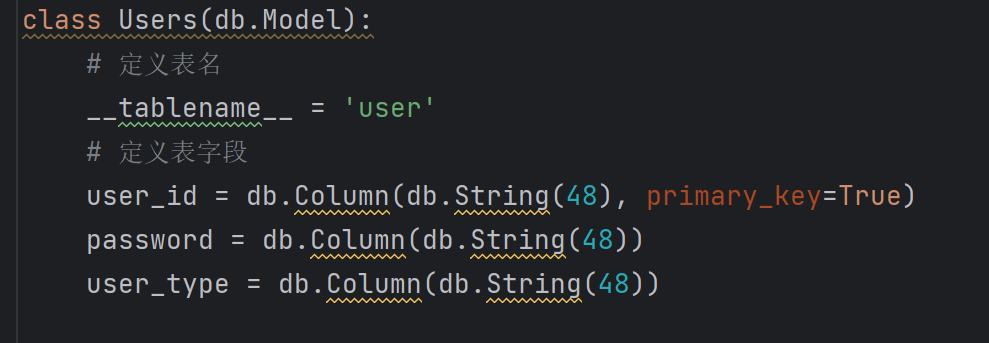
1. **relational schema**



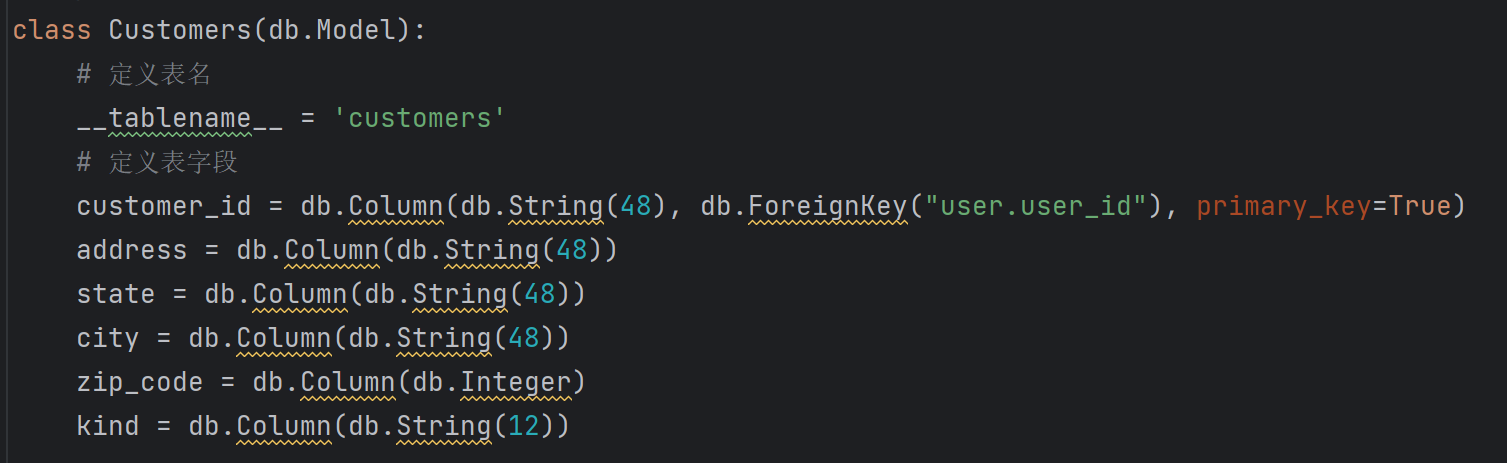
Img4 relational schema

Use the E-R graph, we create the relational schema as the graph shows. The light bulb sign here represents the key; Red values are foreign key. Except the main table required in the project description, we have user, sub transactions, business customers and home customers. The user table is used to uniformly manage all registered user information, including customers and salespersons. Sub transactions are created to allow the main transactions to store multiple goods information at the same time. For example, if we only use the transactions, we will have multiple transaction id with the same “date”, “salesperson id”, “customer id” information for different products. But if we have the sub transactions, we can avoid this duplicate information. Business customers table and home customers table are used to show the information of customers in different types.

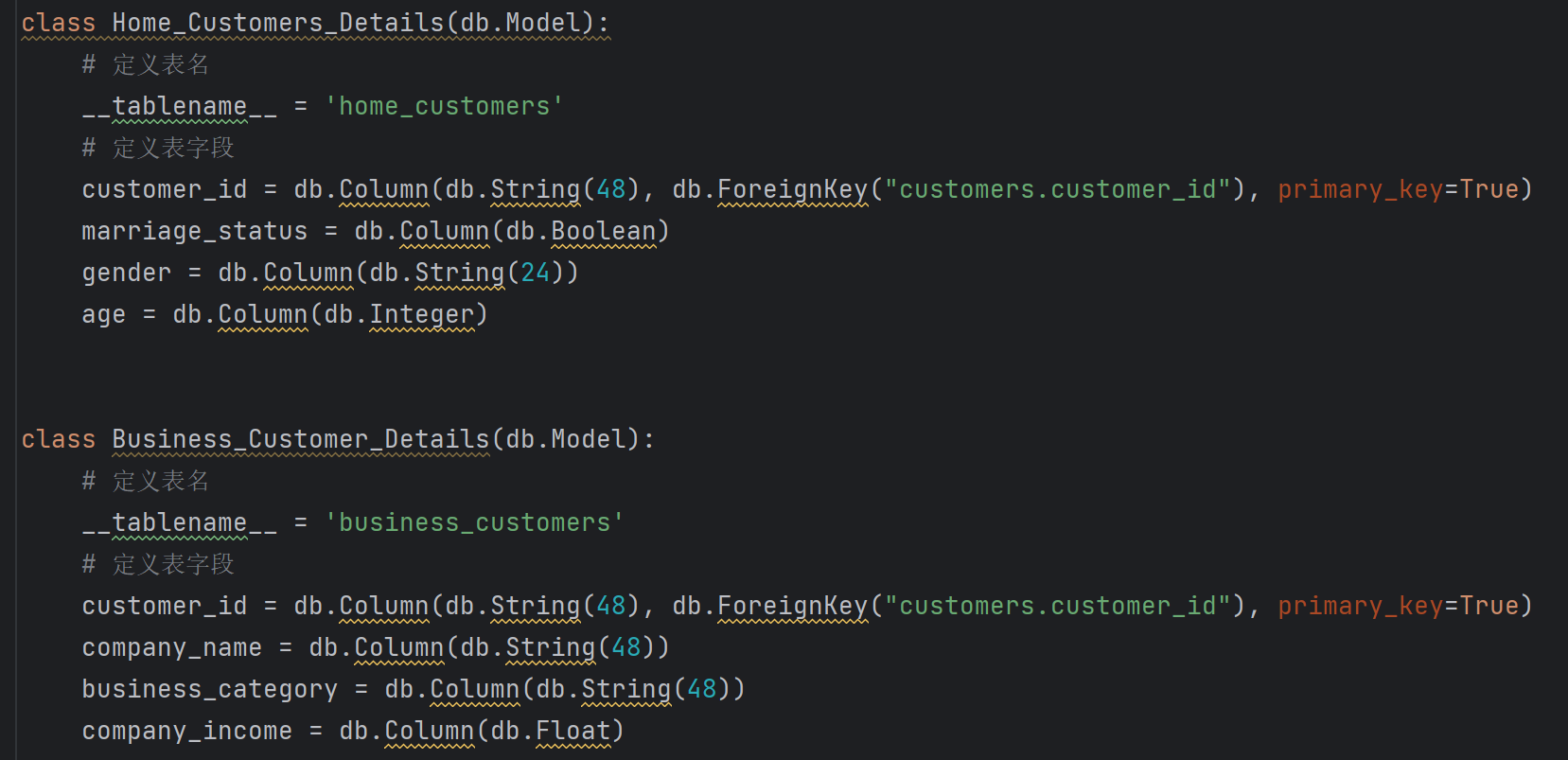
The creation code of the entire database relationship is shown in the figure below:



Img5 user table



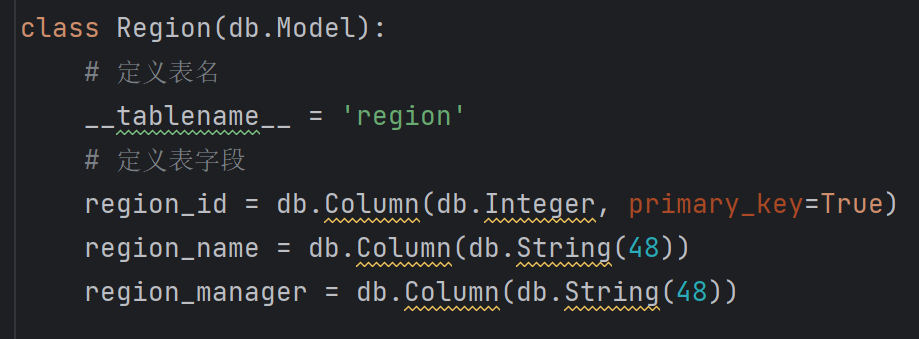
Img6 customer table



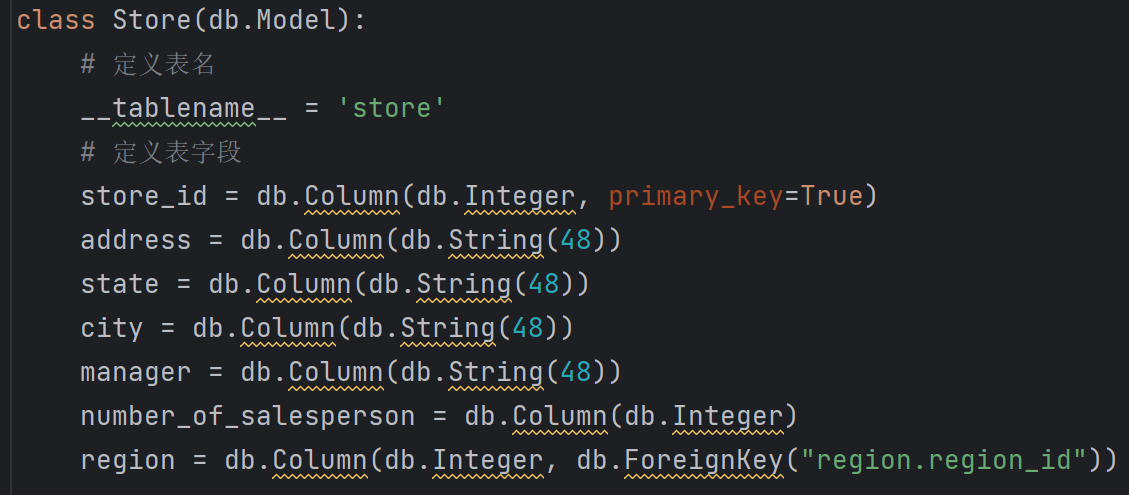
Img7 home and business customer table



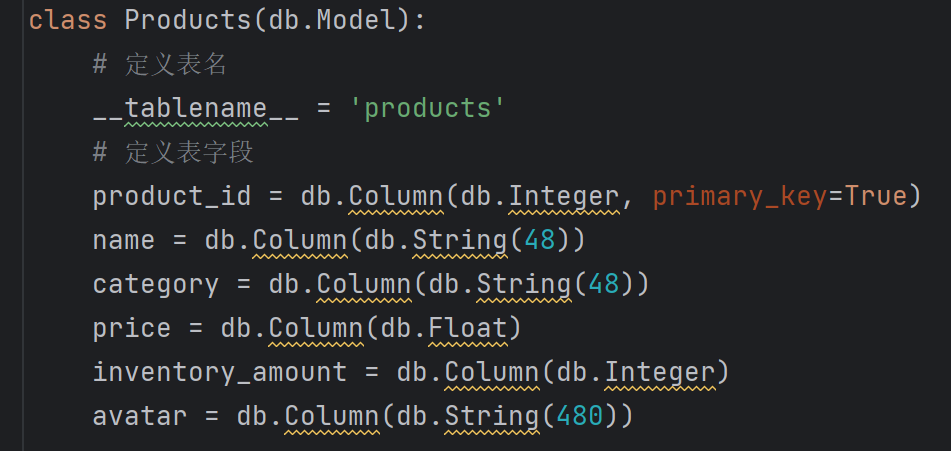
Img8 salesperson table



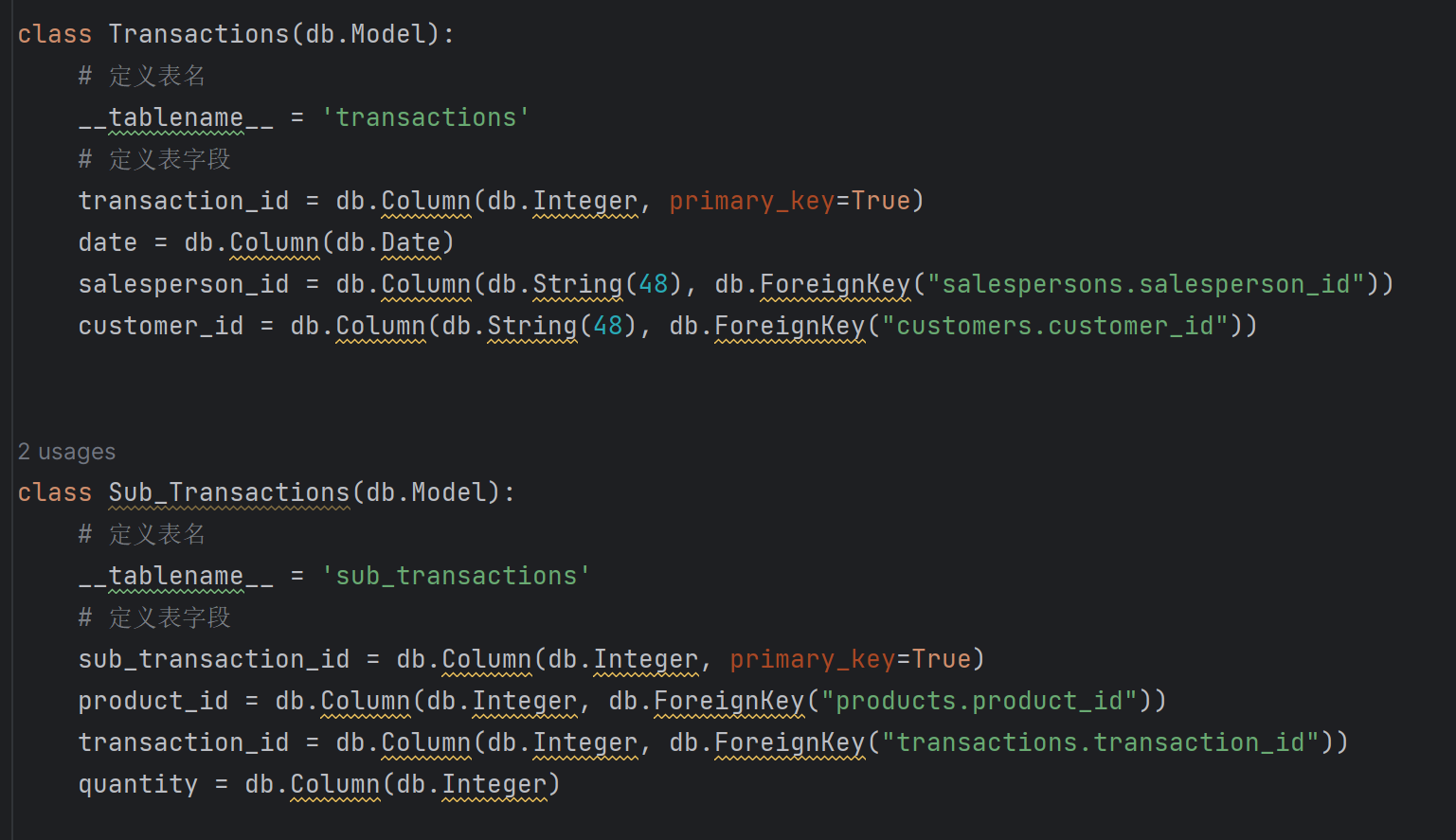
Img9 region table



Img10 store table



Img11 products table



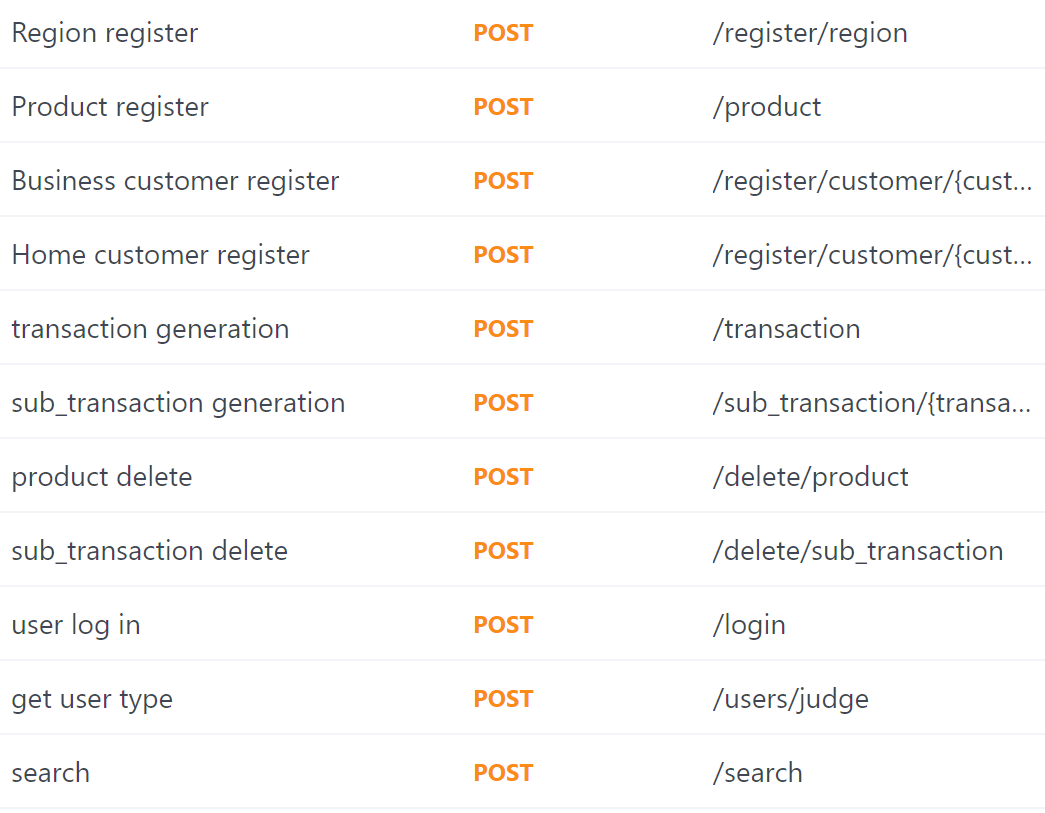
Img12 transactions table

Because the relationship does not contain transitive dependencies and partial dependencies on the primary key, id is the only key for each table. Then the Normal Form is **BCNF**.

1. **API**



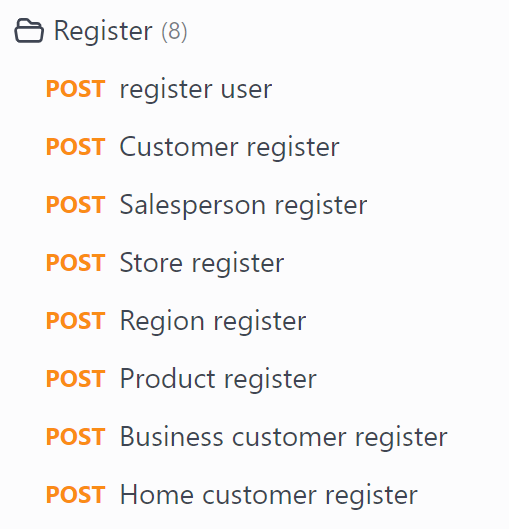
Img13 all the API part 1



Img14 all the API part 2

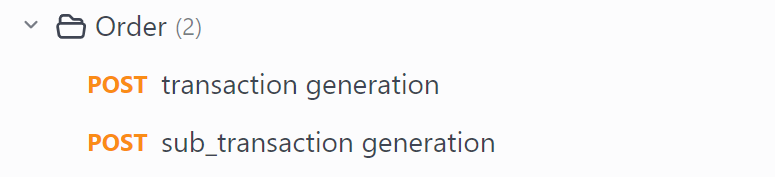
Now let me introduce about the different kinds of interfaces separately.

First the function we want to create is importing data into database. Including user, customer, salesperson, store, region, product business customer and home customer.

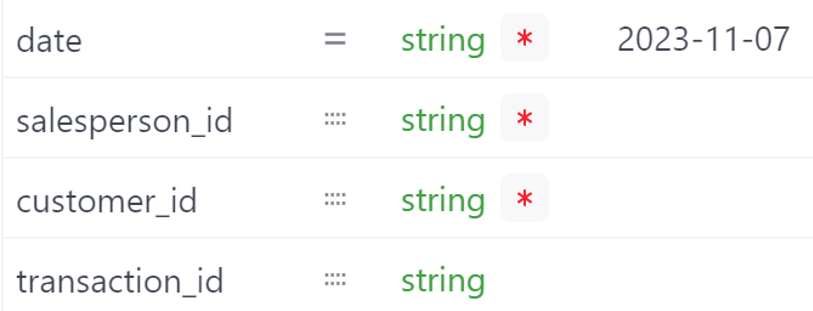


Img15 register interfaces

Then is the main function for our system, the order function. Including transaction and sub transaction. In transaction generation, customer create a transaction and select the salesperson. Then the system will get the date, customer id and transaction id itself.

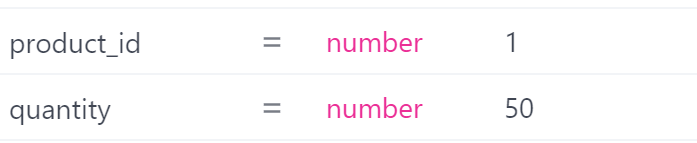


Img16 order interfaces



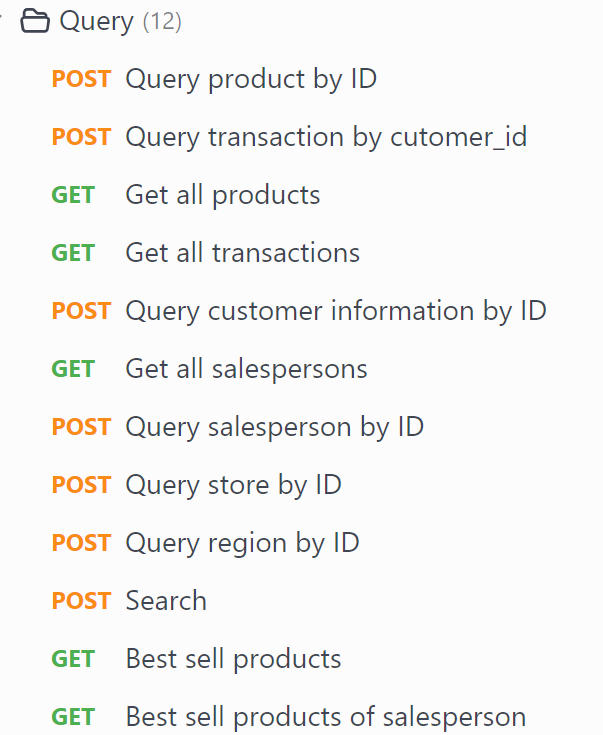
Img17 examples input in transaction

In sub transaction generation, customers will select multiple products and their quantities. The system will create a sub transaction id, then connect the sub transaction to the main transaction.



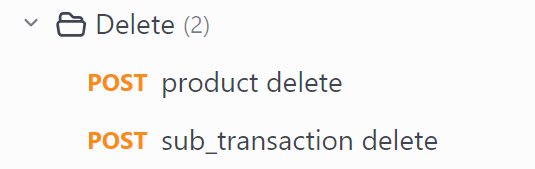
Img18 examples input in sub transaction

Next are query function. Including five ‘GET’ function to get all information from table. And three function to get information by ID. Of course including customer ID and salesperson ID. In particular, the last two query interfaces show two aggregate searches. And the Search interface shows a fuzzy search.



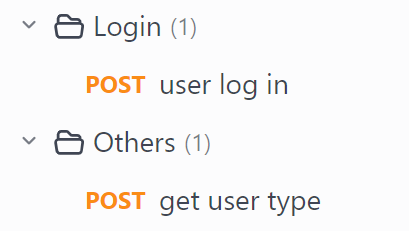
Img19 query interfaces

Then, is delete function. In particular, I will create a function in sub transaction deletion. When all sub transaction under the main transaction bound to the sub transaction are deleted, the main order will also be deleted.



Img20 delete interfaces

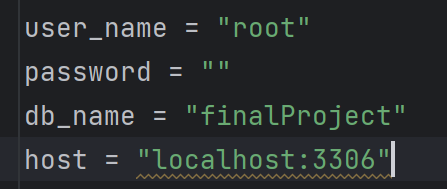
Last are other function. Including log in for customer user and salesperson user. And a get user type function, in order to judge whether user is customer or salesperson.



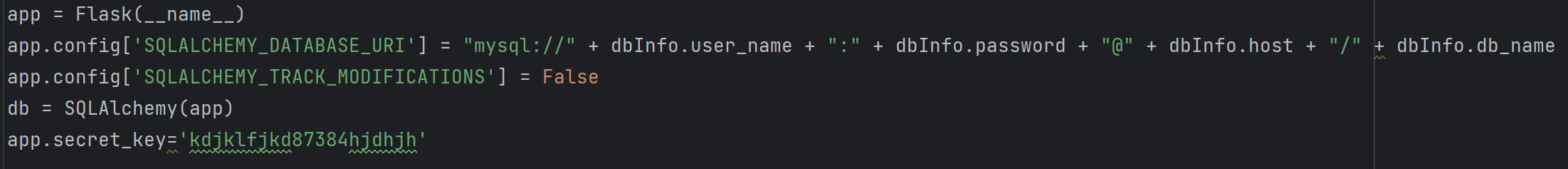
Img21 other interfaces

1. **Backend connection to database**

In backend, I use the python + flask + mySQL to connect to the database. In order to let our front-end use his database, I create a file called “dbinfo.py”, to change the main information to connect to the mySQL.



Img22 dbinfo.py



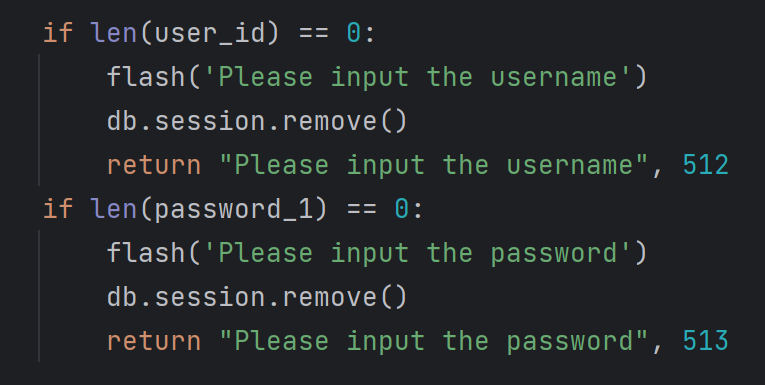
Img23 connection example

**Register:**

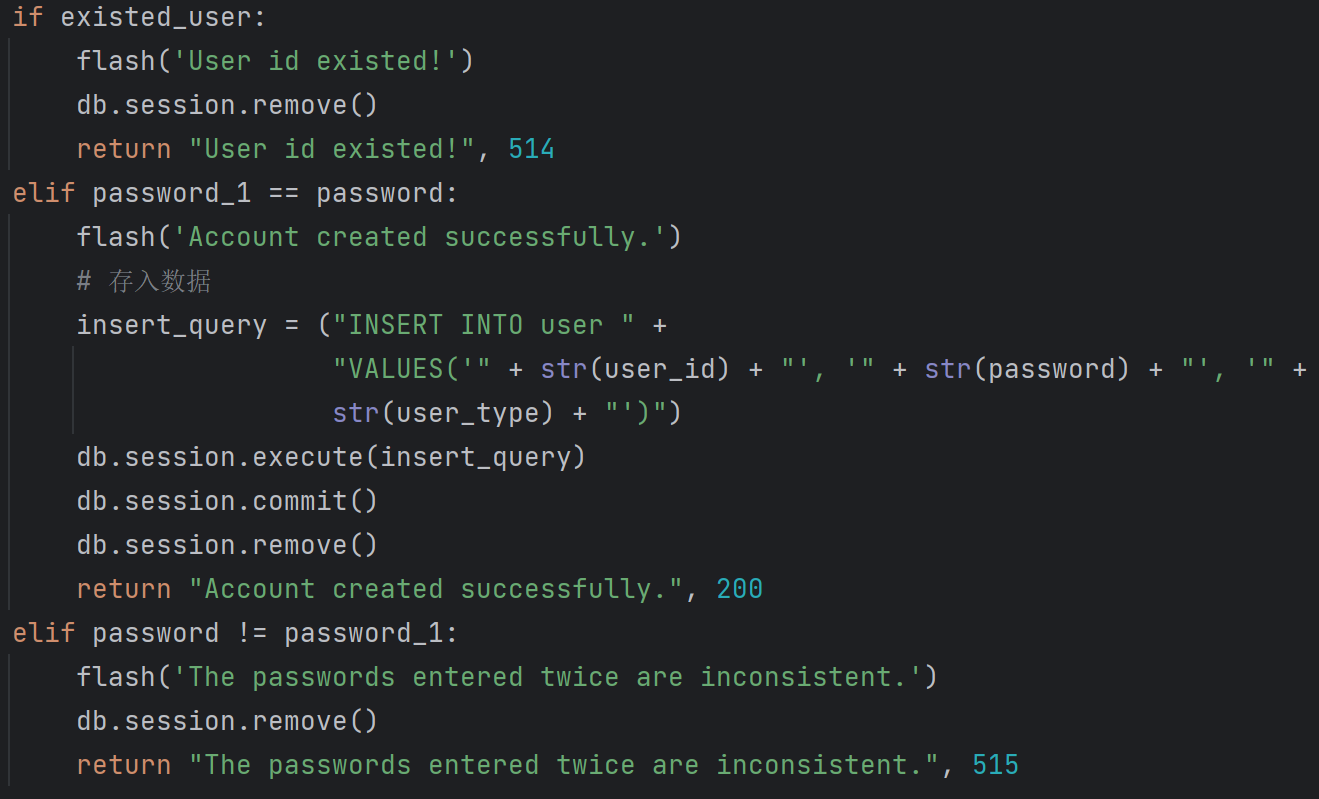
First, we use the request.args.get() to get information from the front-end. Then, using the SQL to write the query statement, and use execute() function to put the SQL query into the mySQL system. After that, we handled some error situations such as inputting null values, invalid input values, etc. At last, we write the input query statement and insert the data into the table in the database.



Img24 register example 1



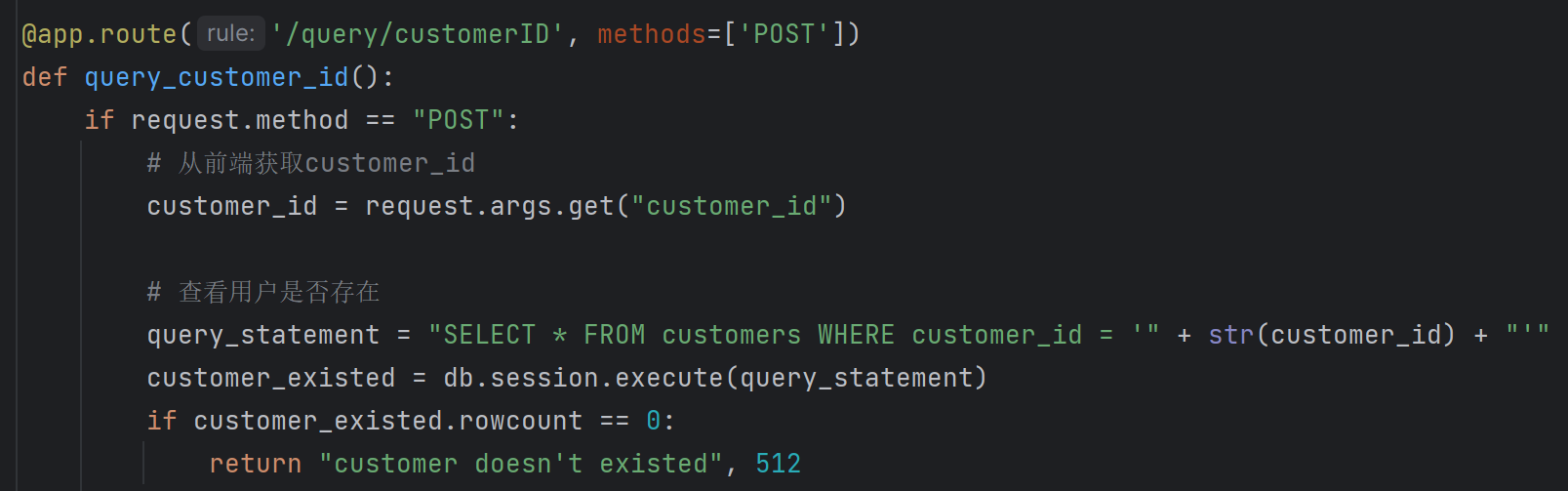
Img25 register example 2



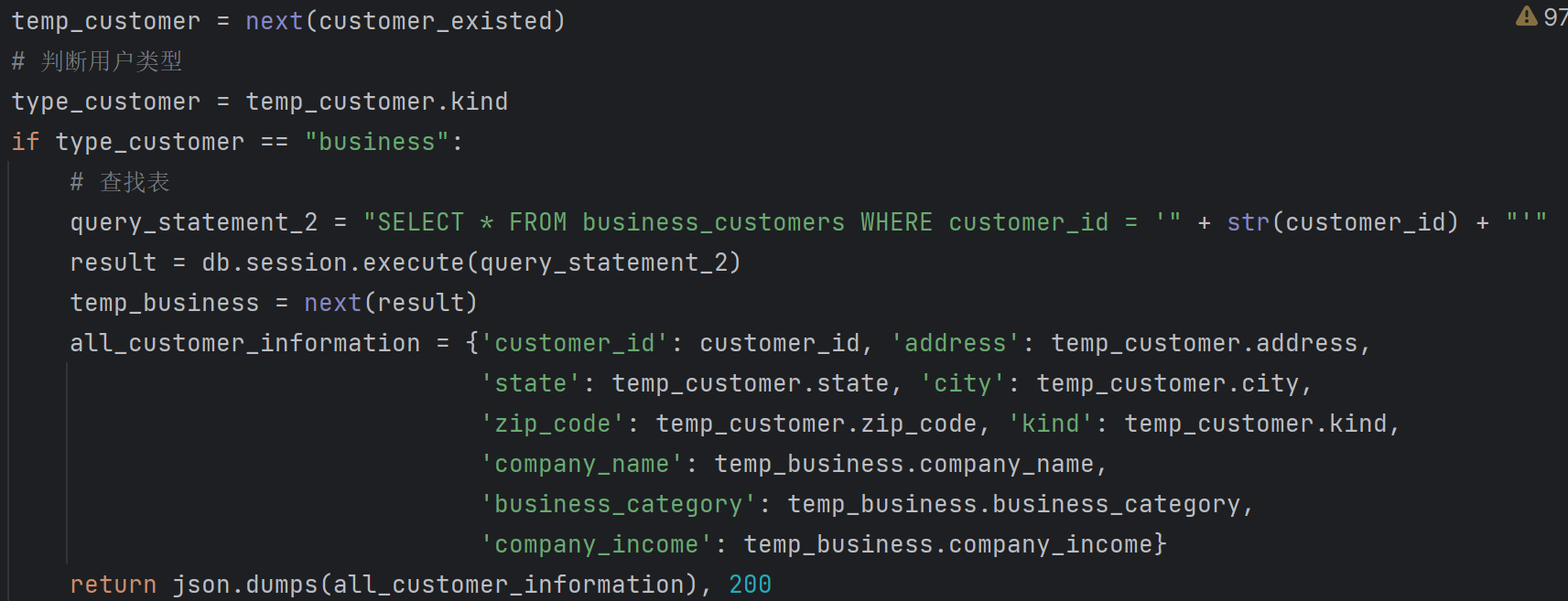
Img26 register example 3

**Query:**

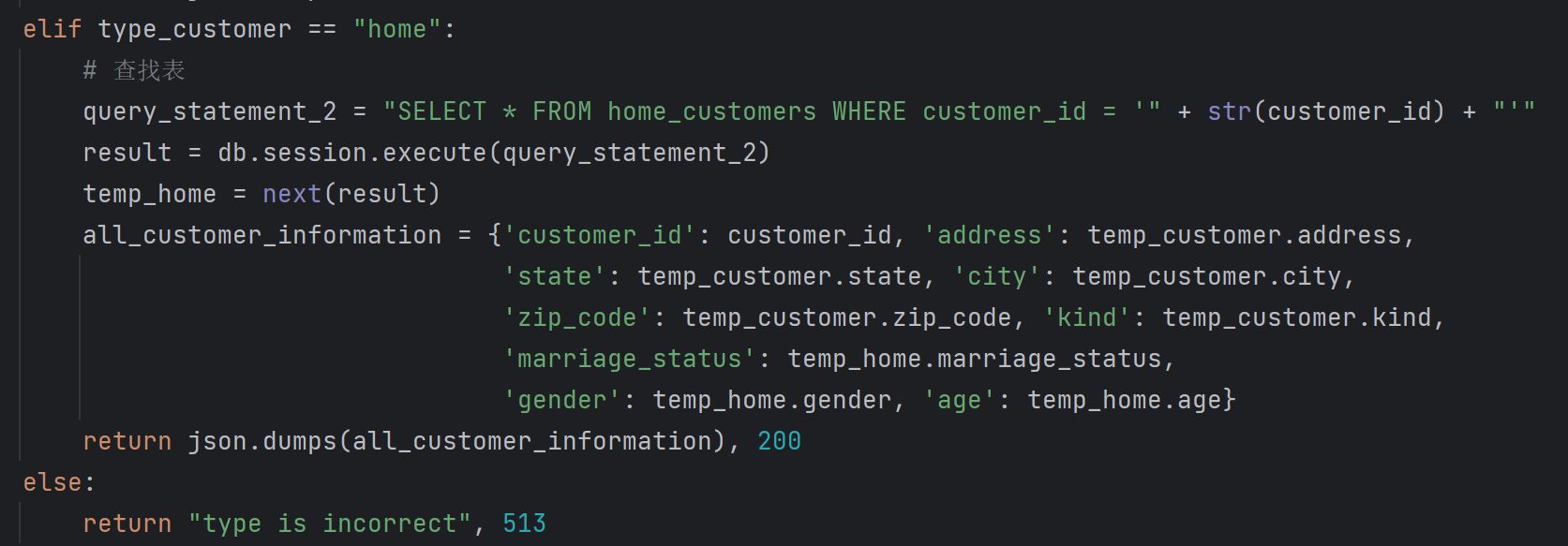
We select an example that use the customer ID to query the information of business customer or home customer. First, I handled some special situations such as inputting null values, user non-existence, etc. If there are no more errors, I first query the information in the customers table. Then I use to kind element to help me judge if customer is a business customer or home customer.



Img27 query example 1



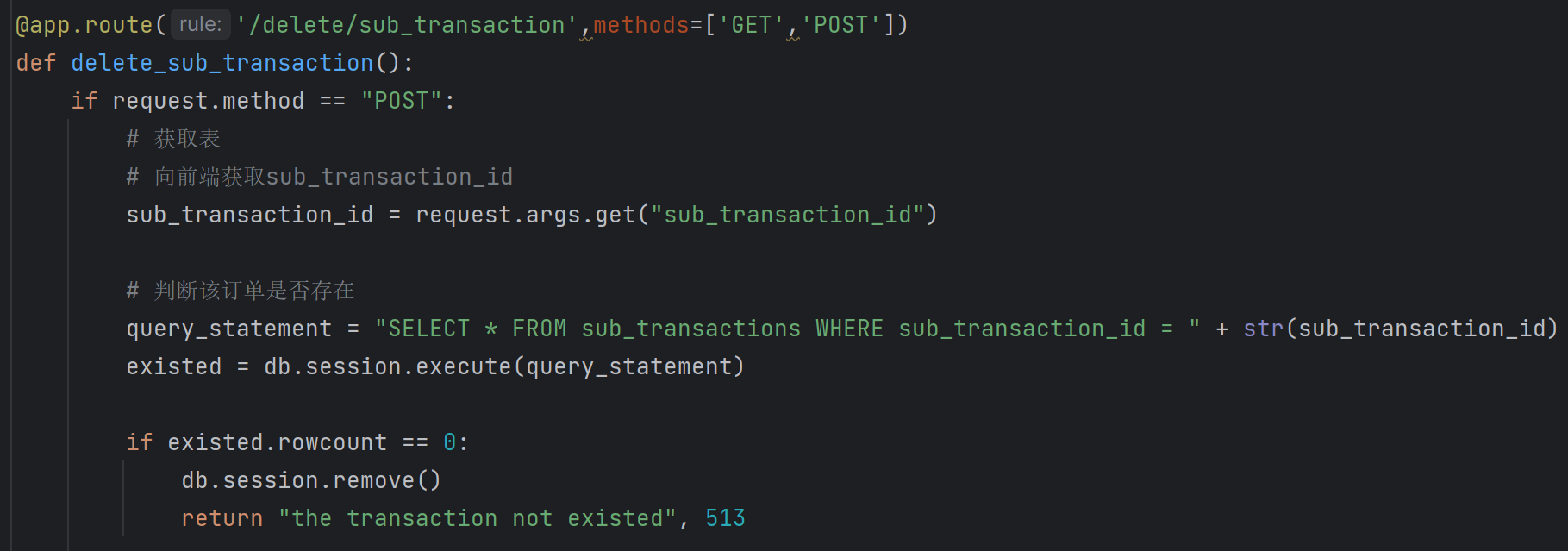
Img28 query example 2



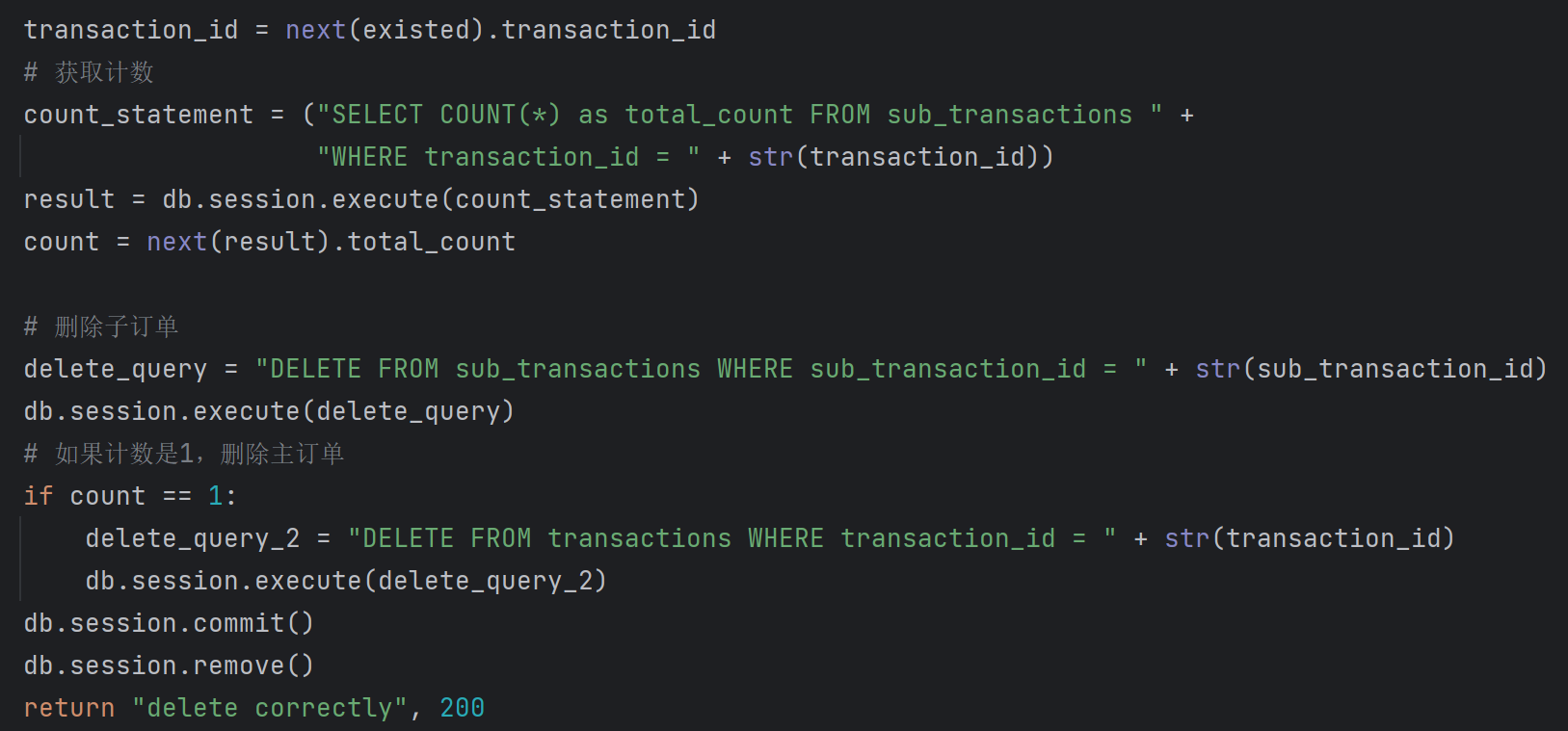
Img29 query example 3

**Delete:**

we choose to show the example of deleting the sub transaction. First, we get the sub transaction id from the front-end, and use the DDL query statement to judge if this sub transaction exists or not. Then we try to delete this sub transaction. If there is no sub transaction with the same main transaction id in the table, we will delete the main transaction.



Img30 delete example 1

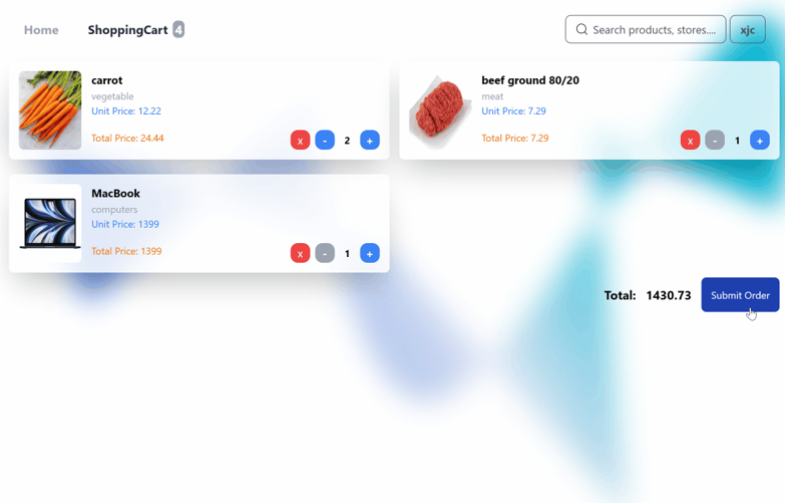


Img31 delete example 2

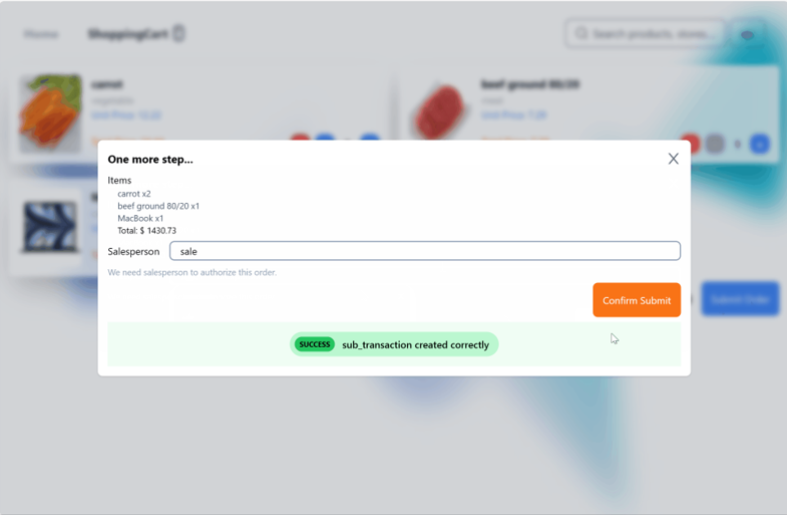
1. **Front-end Design**

**Browse products:**

Customers are able to browse available products in our system, and shopping cart can temporarily store transaction info before submit order. Customer can get information about price, name and other product information.



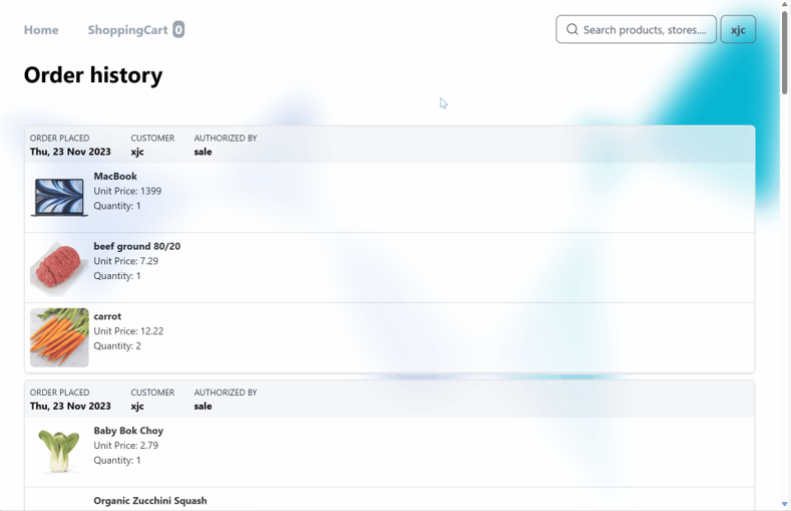
Img32 browse products step1



Img33 browse products step2

**Check orders:**

Customer can check their order history in this page, all required data are fetched from backend, which is an aggression query.

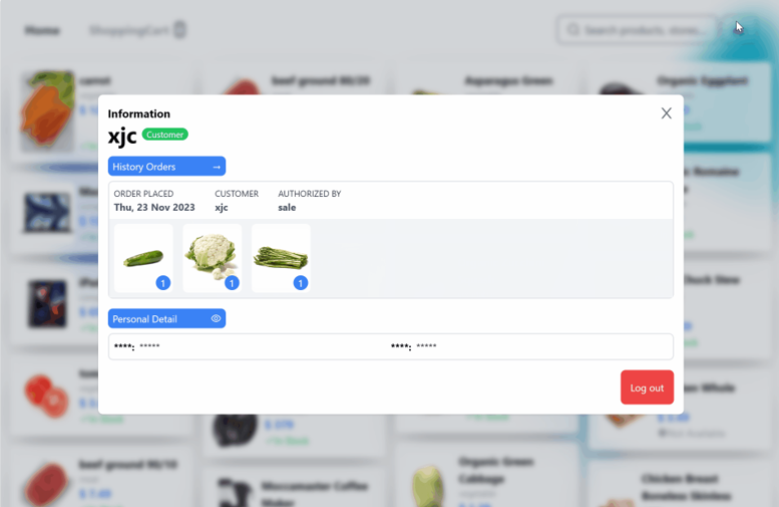


Img34 order history

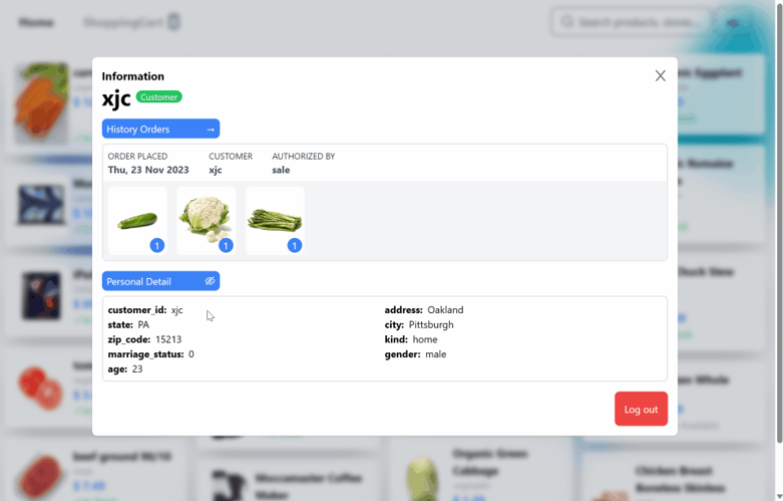
**user information:**

Customers can check their personal information if they have logged in the system. The user identity, e.g., Customer or Salesperson are also marked next to user id.

More detailed information including income, gender etc. can be seen as well.



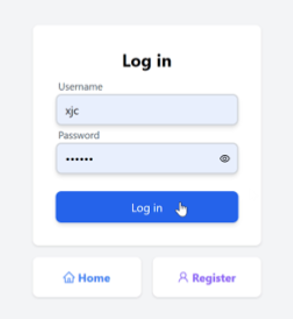
Img35 user information



Img36 user detailed information

**User login:**

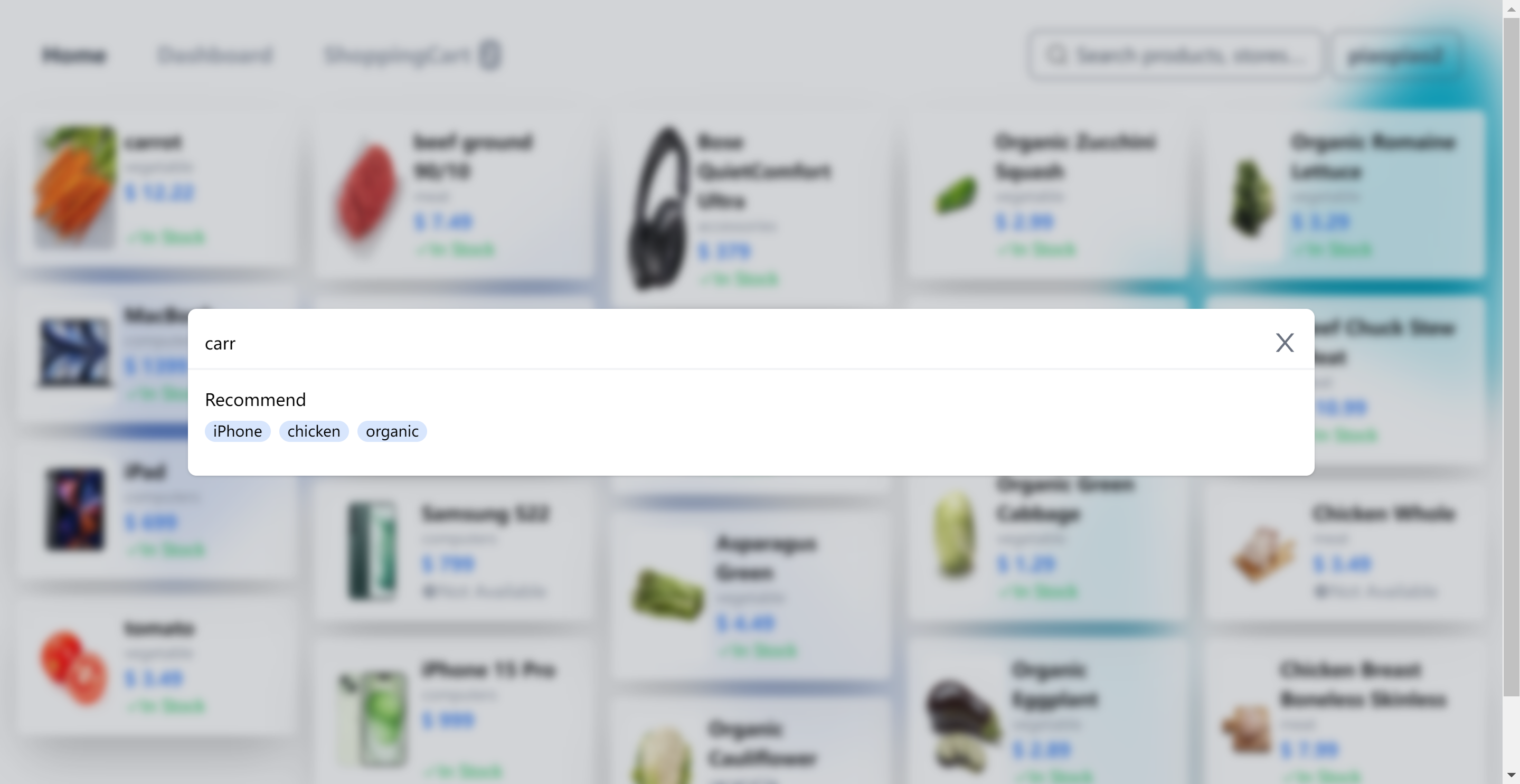
Customers are able to login and register into our system.



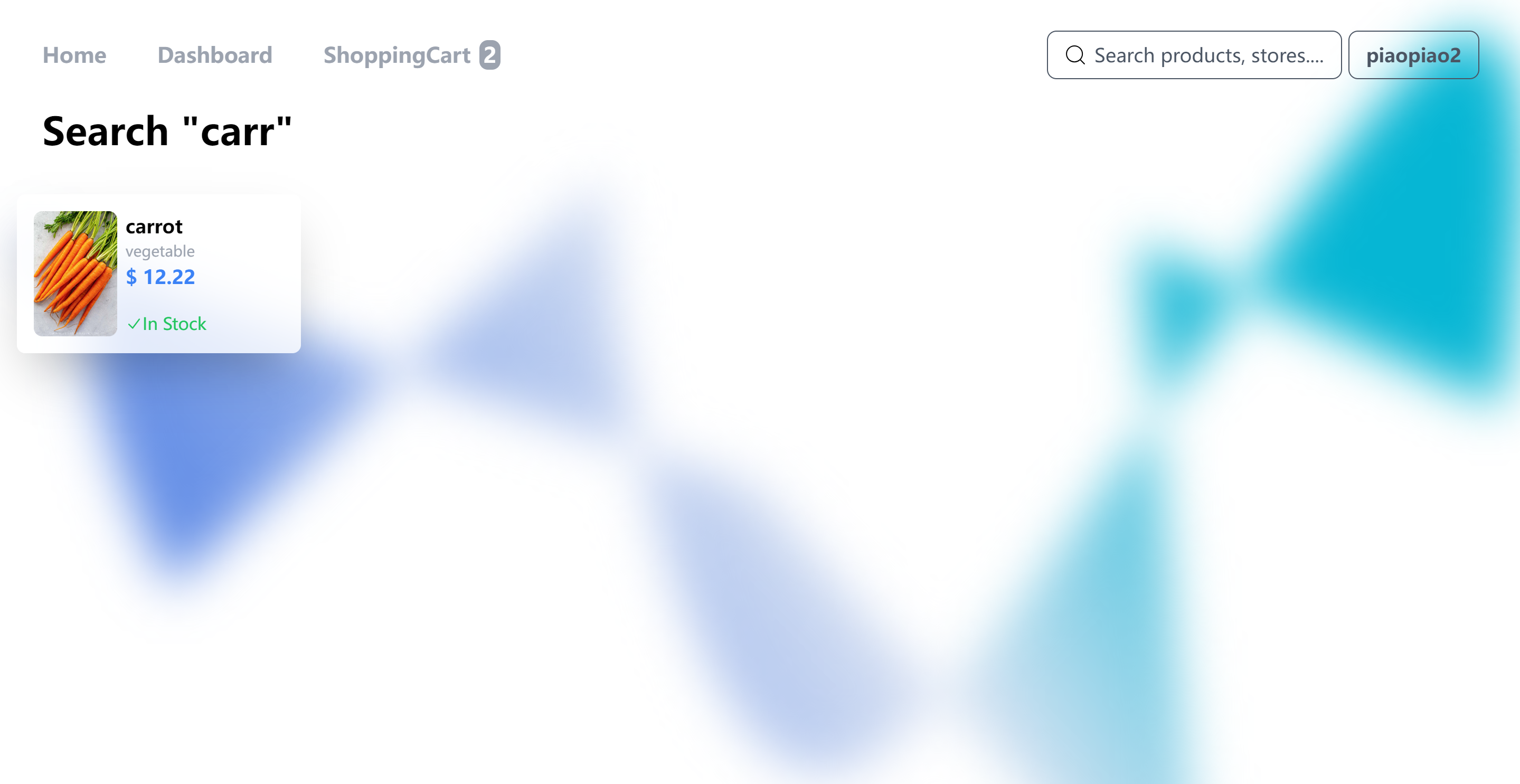
Img37 user login

**Search:**

Users can perform fuzzy searches to find items containing similar words.



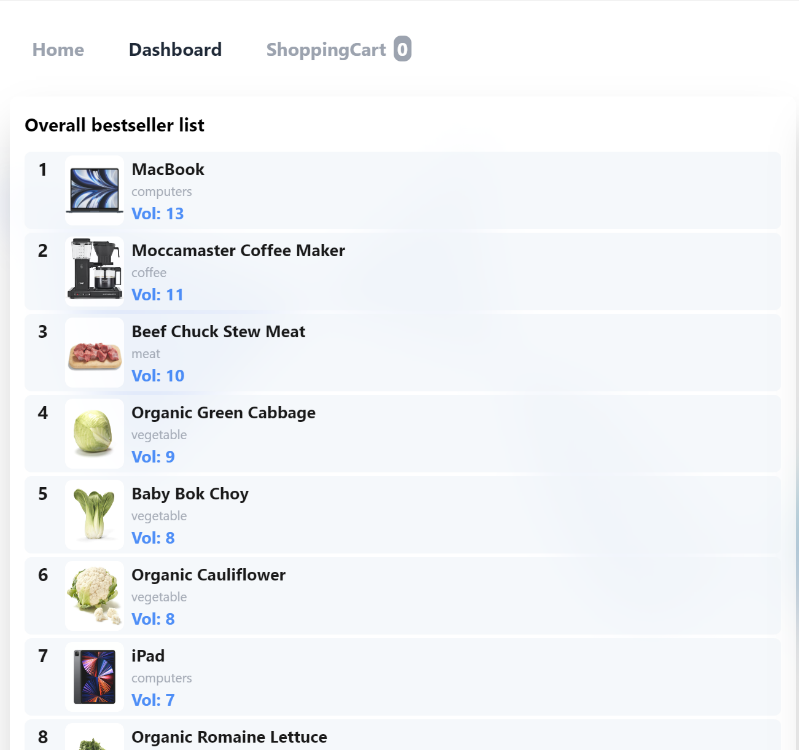
Img38 fuzzy search step1



Img39 fuzzy search step2

**Aggregate search:**

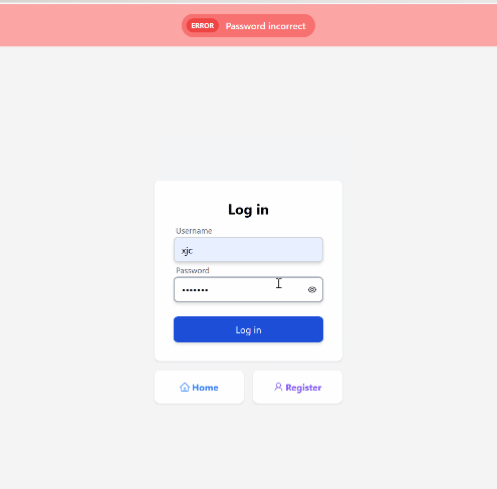
Salesperson can use the aggregate query methods to get the best seller list.



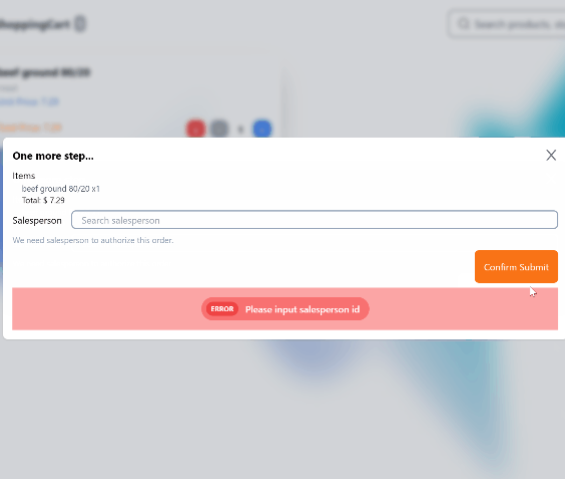
Img40 aggregate query

**Error handling:**

Our system is robust, we handle errors both in front-end and backend.



Img41 error handling example 1



Img42 error handling example 2

**More design details:**

Our frontend design contains more interesting details, for example, you are not able to decrease product number if number is less than 1, the shopping cart page will not show if there’s noting in the cart. Also, is currently number in the inventory is less than 1, we are showing out of stock sign to users.

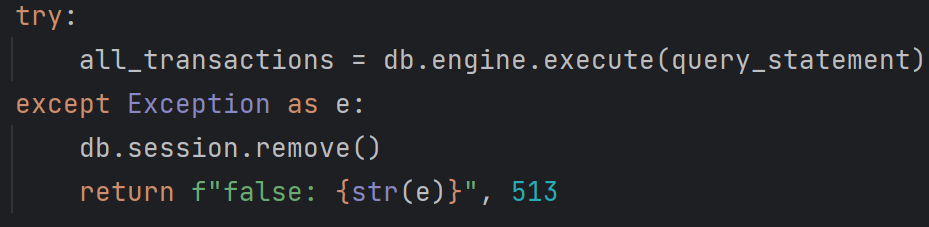
1. **Detectable errors:**

In register, I detect if the input is null value or not.

In particularly, I detect whether the username has been registered. And whether the repeated password is same as the original one.

In delete and query function, I will detect whether the id is existed or not.

And in function that have multiple DDL statement. I will detect the database rollback information if there is any errors occur in mySQL.

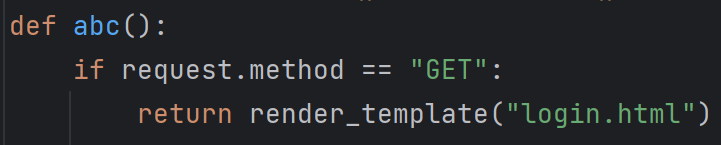


Img43 rollback() detecting example

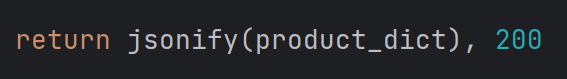
The first error is that the backend doesn’t return in a valid type. Because when using flask to do the backend and front-end, we need to use the render\_template() function to connect the backend and front-end. However, we use other methods except flask in front-end. So that I change to use the jsonify() function to change our information into json type, and pass it with the status code from backend to front-end.



Img44 error message



Img45 error dealing example 1

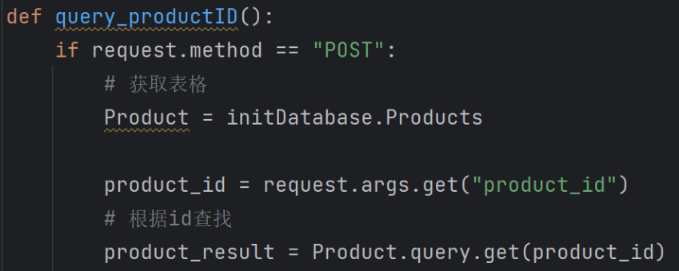


Img46 error dealing example 2

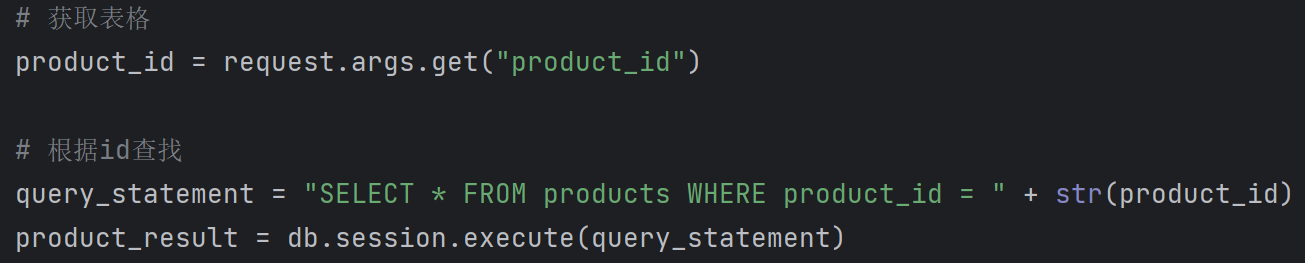
The second error is the QueuePool overflow. I found that when I use SQLAlchemy methods to operate database, when the front-end stop this operation, python won’t disconnect the backend and database. However, when I use the SQL query statement directly, this problem is solved. So, I change the query, insert and delete methods in our code.



Img47 error message 2

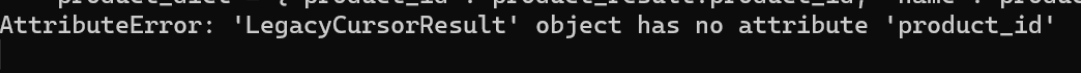


Img48 error dealing example 1

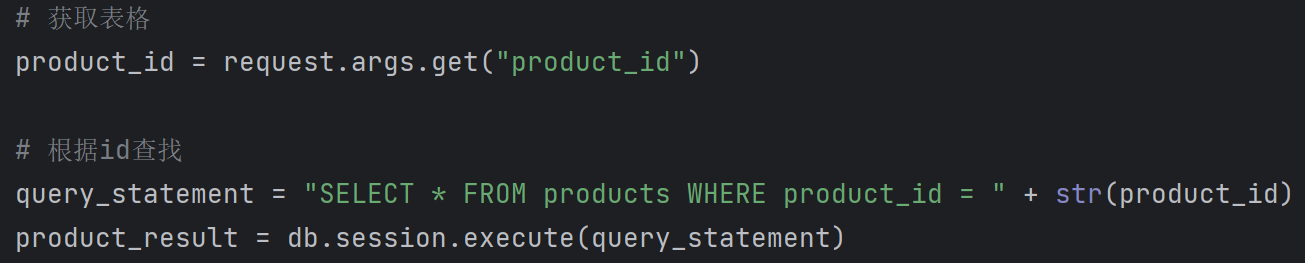


Img49 error dealing example 2

Next, when I try to get the product\_id value in the product result, it shows this error. I checked the information and found out that the ‘LegacyCursorResult’ is a Iterator type for database.

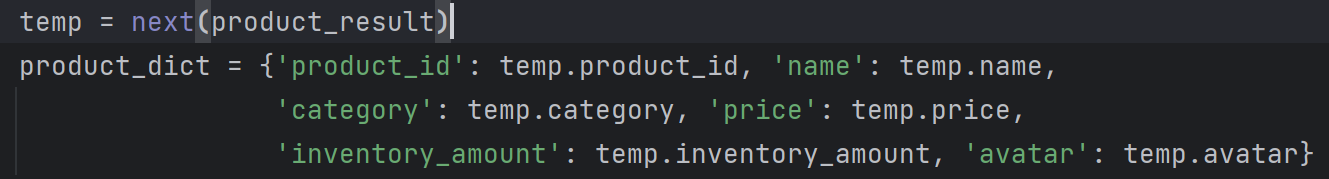


Img50 error message 3



Img51 error dealing example 1

Then I use the function next() to get the data from the iterator to solve this error.

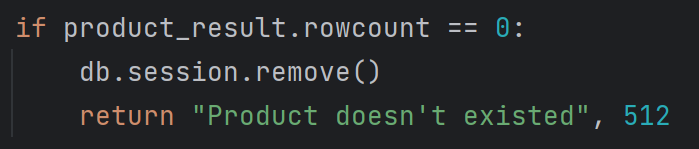


Img52 error dealing example 2

Next error is “raise StopIteration()”. Because I try to use the next() function on a iterator that have nothing in it. So I also change the method to detect the none value to solve this error.



Img53 error message 4



Img54 error dealing example