Spring 2022 CS307 Project2

Part 1-Our team

王宇航 SID 12012208 Lab5	**李玉金 SID 12012225 Lab5**
Paper writing、code	Paper writing、 Discussion of code
Percentage of contribution:50%	Percentage of contribution:50%

Part 2-API Specification

This section describes the purpose, implementation, parameters, and return values of each API. Page request parameters and page return values are described in advanced sections. The source code is not detailed here, but users who need it can see the attachment.

2.0)importData

- This API is used to import initial data with one click.
- Implementation approach:

  The initial data file is read into pandas and converted to its dataframe type. The tables are directly copied to the database using pandas' library functions. The four tables for initial data should be placed at the root of the django_test folder .
- Parameter is web page request, and the return value is web page.

2.1)APIs for manipulating the original data

- These API are used to add, delete, change and check the initial data file.
- Implementation approach:

  These requirements can be handled elegantly by the library functions built into pandas.
- Parameters are web request and others, and the return value is web page.

  The following is the function definition for manipulating the initial data file and parsing the corresponding parameters.

parameter name	parameter meaning
filename	name of the file to be modified
funcType	Type of operating data(add,delete,update,check)
addlist	If the operation type is add, addlist is the row to be added
deleteIndex deleteFlag deleteJudge	If the operation type is delete, the value of deleteIndex is the index of the row to be deleted, the value of deletFlag is the attribute to be determined, and the value of deleteJudge is the condition to be determined. Only the "==" operation is supported.
updateFlag updateJudge updateCol updateData	If the operation type is update and the operation needs to be deleted based on conditions, updateFlag is the attribute to be determined, and updateFlag is the condition to be determined. If the update is not conditional, updateCol indicates the column to be updated, and updateFlag indicates the updated data.
selectFlag selectJudge	If the operation type is select and the operation needs to be deleted based on conditions, select indicates the attribute to be determined, and select indicates the condition to be determined.

Example: If we want to update the values of the name and country fields in enterprise.csv to 'Alcatele' and 'CH', the condition is the row with ID =146.

```
 access_sourceData('enterprise.csv', 'update', None, None, None, None, 'id',
146, ['name', 'country'],['ALcatele', 'CH'], None, None)
```

2.2) StockIn, placeOrder, updateOrder and deleteOrder

- The function of API2-5 is to realize the business logic of purchasing, placing, modifying and deleting orders respectively.
- Implementation approach:
    1. Read the file using pandas. Notice how to parse csv, tsv files.
    2. For the stocking operation, update the product table in the database to increase the current inventory, and update the sale_detail table to increase the counterpart and to reduce the current gross revenue of the goods.
    3. For the placeOrder, order all orders by the time they were signed. Orders are read sequentially. Every time an order is written to the database, we synchronously update the orders and contract tables, and simultaneously update the sale_detail table to increase sales, sales, and profit.
    4. UpdateOrder is basically the same as placeOrder, but note that if the sales volume is 0, the order needs to be deleted instead of the contract .Note that not only the quantity need to be updated, but also the date and other attributes.
    5. DeleteOrder is basically the same as updateOrder, but note that if the sales volume is 0, the order needs to be deleted instead of the contract .
- All API parameters are web page requests and return values are web pages.

2.3)getInformation

- The function of API6 to API13 is to realize the business logic of querying database data.
- Implementation approach:

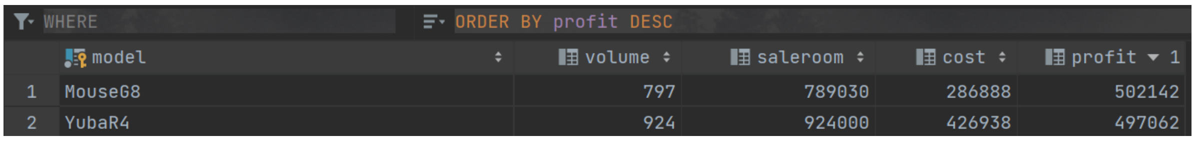
  Since pandas supports native sql statements, we use sql statements for our queries.
- API6 to API14 parameters have parameters representing web page requests, and for each of them return value is web page. API12 has additional parameters that represent the contract to be queried, API13 has additional parameters that represent the order to be queried, and API14 has additional parameters that represent the contract number and order number to be queried.

Part3-Further enhancement to the usability of the APIs

3.1)Bill Module

This project implements a simple billing module corresponding to the sale_detail table in the database .You can calculate the current sales volume, sales volume, cost and profit of each type of product. The form is updated each time an order is purchased, placed, modified, or deleted. (The default is that this project is sold as long as the order is signed) .

In the chart, we can see the types of goods that currently sell with the highest profit.



WHERE ORDER BY profit DESC

	model	volume	saleroom	cost	profit 1
1	MouseG8	797	789030	286888	502142
2	YubaR4	924	924000	426938	497062

3.2)A mechanism to change order status according to time and date

At the same time, the project also realized the automatic update mechanism of the order status, judging the order status according to the actual arrival date .The code logic is as follows:

```
###Each time you log in to the web side, the following program is executed to quickly complete the date comparison using python's datetime library.
cur_time= datetime.date.today()
cur_time=np.datetime64(cur_time)
print('go!!!!!')
for i in range(len(order_fresh)):
        if order_fresh.iloc[i]['lodgement_d']<=cur_time:
            order_fresh.loc[i,'order_type']='Finished'
        else:
            order_fresh.loc[i,'order_type']='UnFinish'
```

At the same time, when we placed the order module, we also completed the judgment logic of the order status, which will not be described here .

Part4-A real back-end server

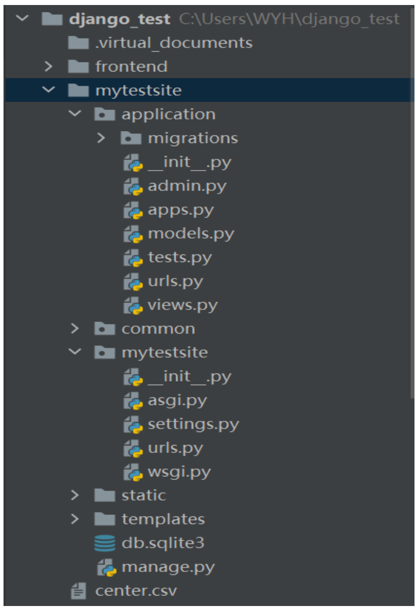
This project uses python Django framework to realize the web backend and configure the database, and uses css, js and other front-end technologies to beautify our front-end html interface, to support users in the web API operations. The following details the process of implementing a back-end server using the Django framework。

4.1)The main Python package version in the project

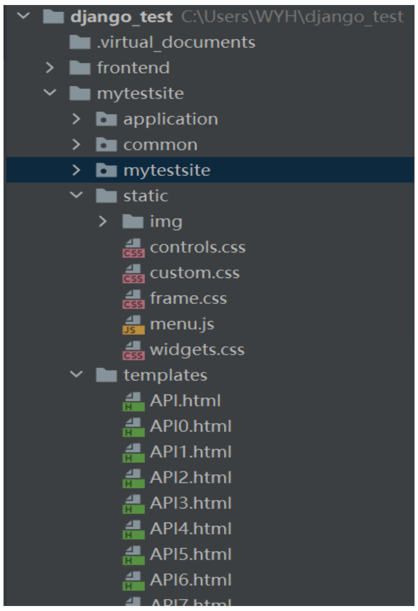
Django's default database is sqlite3, we need to change it to postgresql

Django	django-db-connection-pool	numpy	pandas	SQLAlchemy
4.04	1.10	1.22.2	1.41	1.4.36

4.2)Django framework directory parsing



```
django_test C:\Users\WYH\django_test
  .virtual_documents
  frontend
  mytestsite
    application
      migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py
      views.py
    common
    mytestsite
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    static
    templates
    db.sqlite3
    manage.py
  center.csv
```



```
django_test C:\Users\WYH\django_test
  .virtual_documents
  frontend
  mytestsite
    application
    common
    mytestsite
    static
      img
      controls.css
      custom.css
      frame.css
      menu.js
      widgets.css
    templates
      API.html
      API0.html
      API1.html
      API2.html
      API3.html
      API4.html
      API5.html
      API6.html
```

As shown above, the image on the left shows files on the back end of the Django framework, and the image on the right shows files on the front end of the Django framework。 The following is a breakdown.

The django_test directory is the root directory, and the mytestsite directory under it is the django file directory. Mytestsite contains four folders: application, mytestsite, static, templates, and framework management files: manage.py.

folder name	folder function
mytestsite	The main framework of the project (this project mainly uses setting.py and urls.py files). Settings.py represents the configuration files of the project, such as the database selected, and the APP included in the project. Urls.py represents the routing Settings file. You need to specify the URL path of the view function used by view.py in each app.
application	Functional modules of the project.(The views.py file is used for this project. The model.py file is not used because it does not map tables in the database to django supported objects)View. py is a view function that accepts user requests and returns the corresponding web page to the user after parsing the request. Urls.py represents the child routing file, and the corresponding parent routing file is urls.py in myTestsite.
static	Contains the images used for the project's web page, a series of components used to render the web page, scripts and other files. These are static files. The static file path must be specified in setting.py in mytestsite.
templates	This is the web page template of the project. It contains the html template corresponding to each function of the project) Test.html is the main page, from which the rest of API.html jumps.

4.3)View functions in detail

Here we use the API13 example to describe the arguments and return values of the view function in detail

```
def API13(request):
    if request.method == 'POST':
        name = request.POST.get('name13')
        contract = api13(name)
        return render(request, 'API13.html', {'contract_num': contract[0],
'con_final':
        contract[1]})
```

As shown in the figure above, all API parameters are webpage requests, and the return value is the corresponding webpage (for static webpage, there is no need to return the dictionary; for dynamic webpage, webpage content parameters are determined by the dictionary in the return value). All APIs of this project adopt this design, and other APIs will not be described again.

4.4)How to interact with users

In html templates, we can interact with the user by adding button components and text box components.

Here we use API13 from the main page of test.html as an example and illustrate the code with comments.

```
</div>
    <!-- Render component with button name API13. -->
    <div class="control-group">
    <button class="custom-button-flat "><img src="../static/img/collection.png">
<span>API13
    getContractInfo</span></button>
         <!-- The request type is post and the corresponding url of webpage is
API13. -->
        <form method="post" action="{% url 'API13' %}">
            <!--Prevents csrf attacks and sends the token information in the
cookie of the client to the server. If the request does not contain the token,
the request is invalid.-->
            {% csrf_token %}
            <!-- Provides a text box for user input, defines a submit button, and
jumps to API13. -->
              <input class="custom-textbox stretch-on-mobile" type="text"
name="name13"
                placeholder="CSE0000219">
              <input type="submit"  value='commit' style='font-size:5px'
                onclick="window.location.href='../API13'"/>
        </form>
</div>
```

Part5-Database connection pools

In the case of frequently creating and clearing database connections, the consumption of database resources is undoubtedly huge. The solution is to create a pool of database connections and create a specified number of database connections when the project is created. Then replace creating a database connection with getting a connection from the connection pool. Replace clearing connections with connecting back to the connection pool. Thus we reduce the consumption of database resources.

Due to the fact that django does not support database connection pooling, you need to use a third-party library to configure database connection pooling. This porject uses django-db-connection-pool. You can run the following command to install it.

```
pip install django-db-connection-pool
```

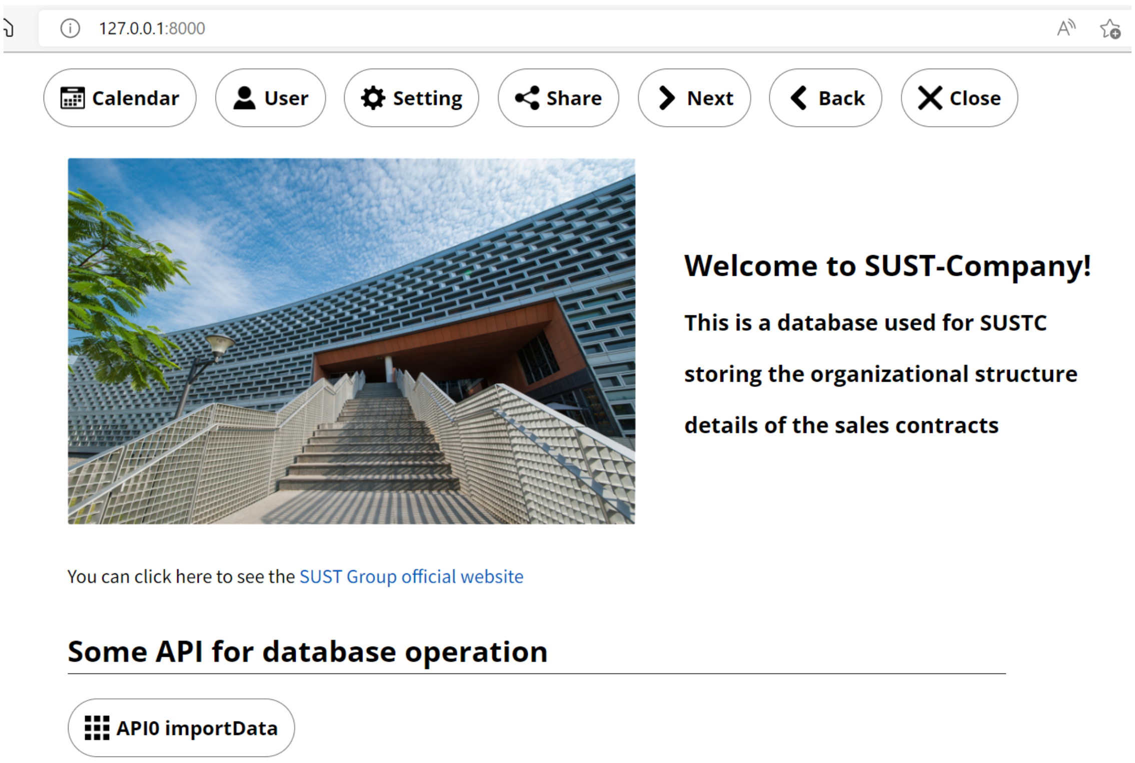
Then, in Django's Settings file, modify the database parameters.

```
DATABASES = {
    'default': {
        # 'ENGINE': 'django.db.backends.postgresql',
        'ENGINE': 'dj_db_conn_pool.backends.postgresql',
        'POOL_OPTIONS': {
            'POOL_SIZE': 100,
            'MAX_OVERFLOW': 50
            #Maximum number of connections is POOL_SIZE+MAX_OVERFLOW=150.
        },
    }
}
```

Part6-A usable and beautiful GUI

In this project, we use CSS style and JS scripting language to beautify our html template. At the same time, we display the results of the query in the form of tables to make the data more intuitive.

- Homepage:



Welcome to SUST-Company!

This is a database used for SUSTC

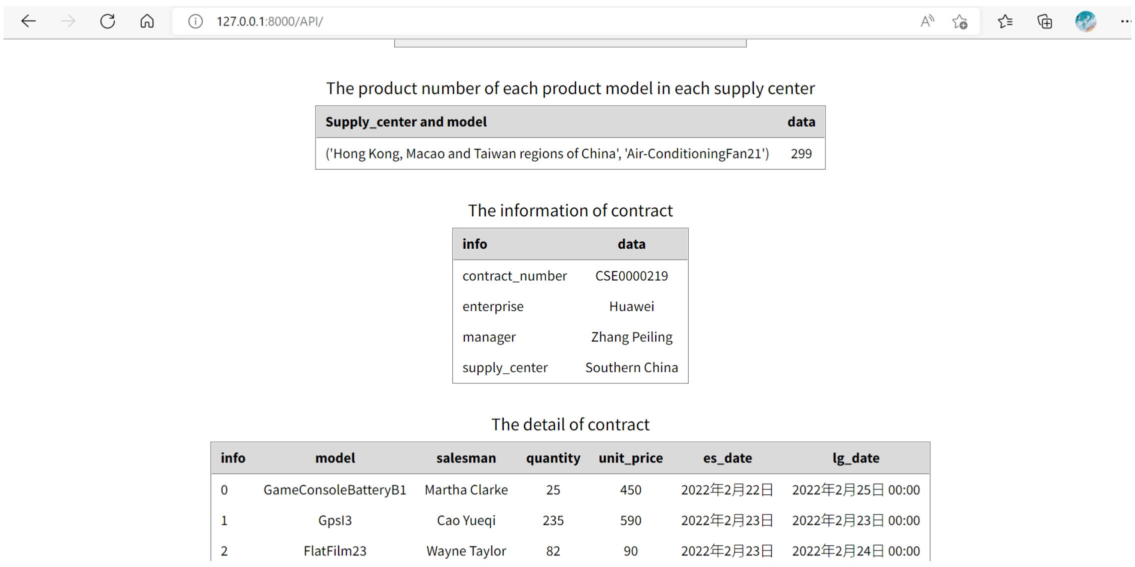
storing the organizational structure

details of the sales contracts

You can click here to see the SUST Group official website

Some API for database operation

- Use the form of table to show the data, which is more intuitive and clear.



The product number of each product model in each supply center

Supply_center and model	data
('Hong Kong, Macao and Taiwan regions of China', 'Air-ConditioningFan21')	299

The information of contract

info	data
contract_number	CSE0000219
enterprise	Huawei
manager	Zhang Peiling
supply_center	Southern China

The detail of contract

info	model	salesman	quantity	unit_price	es_date	lg_date
0	GameConsoleBatteryB1	Martha Clarke	25	450	2022年2月22日	2022年2月25日 00:00
1	Gpsl3	Cao Yueqi	235	590	2022年2月23日	2022年2月23日 00:00
2	FlatFilm23	Wayne Taylor	82	90	2022年2月23日	2022年2月24日 00:00

Part7-Database index

Creating an index for the corresponding field of a table in the database helps to speed up its search speed. Accordingly, as the price of accelerating the search speed, the insert, update and delete speed of the table will be slower. Therefore, we created the corresponding index for the following table after calculating the frequency of adding, deleting, changing and checking of all APIS on the table.

```
create index product_index on product using btree(model);
create index staff_index on staff using btree(type);
create index order_index on orders using btree(contract_number,salemans_id);
create index sale_detail_index on sale_detail using btree(volume);
create index contract_index on contract using  btree(c_number);
```

Part8-Preliminary exploration of defending against CSRF attack

When we create html templates for web pages, we use text boxes to receive user input. However, when we clicked submit, the website displayed the 403 Forbidden error message. After checking, we found out that this is a Django mechanism to defend against CSRF attacks. We need to insert this line in the corresponding component code to pass Django verification.

```
{% csrf_token %}
```

Therefore, we do a preliminary study on the attack mode and preventive measures of csrf attack.

8.1)Meaning of a CSRF attack

CSRF stands for Cross-site Request Forgery, which is another way of putting it: an attacker steals your identity and sends a malicious Request in your name that is perfectly legitimate to the server. Such as sending emails in your name, stealing your account, or even transferring money. Therefore, CSRF attack is very harmful.

8.2)Mechanism of CSRF attacks

First of all, we need to understand what cookies are. Simply speaking, cookies are the information about users stored by the server on the client for identifying users. Each time a user visits the website, the cookies of the user client will be carried by the server. The server can judge whether the user has the permission to perform certain operations according to this information. If the user has insufficient permissions, the request will be blocked by the server. CSRF attacks use cookies of others to verify permissions when the attacker does not have permissions, so that the attacker can conduct illegal operations without the knowledge of the attacked.

8.3)Defence against CSRF attacks

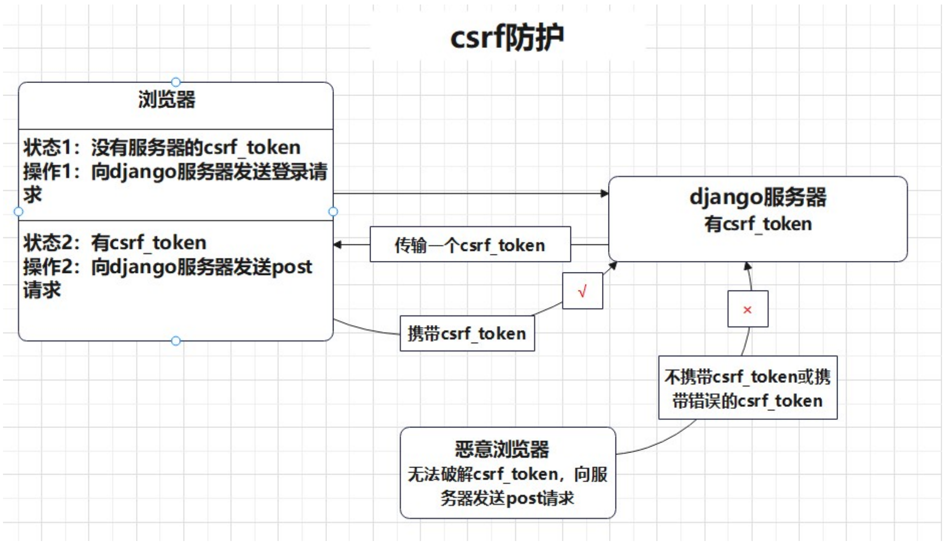
The reason why CSRF attack can be successful is that the hacker can completely forge the user's request, and all the user authentication information in the request is in the cookie, so the hacker can directly use the user's own cookie to pass the security authentication without knowing the authentication information. The key to defending against CSRF is to include information in the request that a hacker cannot forge.

Django takes the approach of adding a randomly generated token to an HTTP request in the form of a parameter. Each time a user initiates a request, not only must he/she bring his/her own cookies, but he/she must also carry the token. The server side will establish an interceptor to verify the token. If there is no token in the request or the token content is incorrect (different from the token on the server side), the request will be rejected as a CSRF attack.

When checking cSRFtoken, only secret is compared, not the whole token. The first 32 bits of the token string are salt, followed by the encrypted token. Only secret can be decrypted through salt. This allows the form's token to be refreshed continuously, while the client's token remains unchanged for a longer period of time.

```
###In checking the webpage, we found that {% csrf_token %} was changed to the following string, value is the token.
<input type="hidden" name="csrfmiddlewaretoken" value="xdjfCeW13mozq3VhrTyyZ5AtzC13etB0ipsnfpp2vfChk5Fpp1LjBQIRAIFj9Yc4">
```

Here we draw a flow chart to visually reflect the flow of CSF protection.


csrf防护
浏览器
状态1：没有服务器的csrf_token
操作1：向django服务器发送登录请求
状态2：有csrf_token
操作2：向django服务器发送post请求
django服务器
有csrf_token
传输一个csrf_token
√
携带csrf_token
×
不携带csrf_token或携带错误的csrf_token
恶意浏览器
无法破解csrf_token，向服务器发送post请求

Part9-Conclusion

In this project, our team cooperated sincerely. We completed 13 API designs and a one-step design for reading or output data. We then further refined the API design by designing billing modules and mechanisms to change order status based on date and time, allowing it to accept requests with more flexible parameters. Next, we encapsulated and implemented a real back-end server, using database connection pooling to construct a useful and elegant interface for rendering data, and using indexes to speed up the manipulation of tables. Finally, we preliminarily explore the attack mode and preventive measures of CSRF attack.

At the same time, we also encountered many difficulties in this project

- The configuration of the Django environment (it is recommended to specify a stable version when installing the package. If you do not specify the latest version, it may not be compatible with other libraries).

- The pandas merge function is used to manipulate the pandas data. The pandas merge function is used to manipulate the pandas data. Merge functions are basically the same as join functions in SQL statements. However, after merging, tables are not sorted in the same order as before. The order module of this time is very sensitive to the time field (inventory needs to be updated in real time according to the order), but the time field of the data of this time is not accurate enough, and the minimum unit is day. In a series of operations, the relative order of the same day cannot be changed. If the time field is accurate enough, we can convert it to a timestamp for sorting. So in the end, I used pandas' join function for time-sensitive tables, without using merge.

- For scheduling pandas by time, we need to specify the sort type as mergesort. The quicksort is unstable
- For HTML, CSS, Web framework these things, because it is the first contact, so be sure to learn more and ask more, not online search, as long as you have patience, problems can be solved. During command line operations, do not be afraid of red error messages. Read the meaning of error messages first and then refer to the corresponding solutions.

Finally, thank you for making the user **yenchiah** page template design train of thought, Thanks to the teachers and teaching assistants for their efforts in this semester, thanks to anyone who has given us any help in this project!