



THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC 3170

DATABASE SYSTEM

PROJECT REPORT

Report for CSC3170 Course Project Option 1 (Enhanced)

Group:

AMDIVIA

Project Date:

November 25

Contents

1	Introduction	2
2	Team Members	2
3	Historical Progress	2
3.1	Trading System Front-End	2
3.2	Trading System Back-End	4
3.3	Banking System	5
3.4	Target Analysis	6
4	Program Design	7
4.1	ER Diagram	7
4.2	Relation Schema	9
5	Functionality Implementation	10
5.1	Front-End Implementation	10
5.2	Back-End Implementation	12
5.3	Data Analysis Implementation	22
6	Conclusion	25
7	Contribution	25

1 Introduction

This is our implementation for the course project of CSC3170, 2022 Fall, CUHK(SZ). We are going to design a system that provides users with a convenient platform to purchase chips. It is committed to reducing users' money and time costs and improving the production capacity and efficiency of the chip industry. Our project can bring a qualitative leap to the industry.

To accomplish our vision, we implemented three separate systems. First, we will provide users with a simple, elegant, and easy-to-use web interface. Users can obtain intimate and customized services on the website. Besides, our accurate and powerful data analysis system can provide a solid technical foundation for this service. In addition, our independent and safe banking system guarantees the stability and timeliness of transactions.

2 Team Members

Our team consists of the following members, listed in the table below (the team leader is shown in the first row, and is marked with a flag behind his/her name):

3 Historical Progress

3.1 Trading System Front-End

We designed and produced a robust, elegant, and easy-to-use front-end interface. Aiming to provide the user with the best user experience and productivity.


Student ID	Student Name	GitHub Account (in Email)	GitHub Username
120090175	彭乔羽 	120090175@link.cuhk.edu.cn	KillerPQY
120090661	邓毅轩	939681959@qq.com	wek-deng
120090443	周昱潇	120090443@link.cuhk.edu.cn	axbybgl
120090737	刘宇轩	120090737@link.cuhk.edu.cn	LIUAnakin
120040088	王熹	120040088@link.cuhk.edu.cn	iamgeorge
120090721	熊一鸣	120090721@link.cuhk.edu.cn	zksha
120010035	黄熹正	120010035@link.cuhk.edu.cn	TheBigDoge
120010061	林泽昕	120010061@link.cuhk.edu.cn	webDrag0n
120090638	孙绍强	120090638@link.cuhk.edu.cn	ShaoqiangSun
120090673	曲恒毅	qulend@163.com	HomuraCat

Figure 1: *Team Members* Diagram

The front-end interface has mainly four pages:

1. Login page
2. Order list page
3. Order making page
4. Factory info page

On the login page, the user could log in with his user id and password. After login, we can link the user id with the orders made by the user and fill up the order list page. The order list page is a page that shows all the processing orders. If the user wishes to make a new order, he/she could first go to the factory info page and look up detailed information about a specific factory. Including name, location, the number of machines, types of chips that could be produced by the machines in the factory, and more. With the knowledge of factory details, the user could select from a range of dropdown lists. The user-designed chip ordering plan will be sent to the back-end data analysis module and receive a score towards

the plan. If the plan can be executed, the front-end interface will tell the user to confirm the plan and place the order; then, the user will go back to the order list page. Otherwise, the user will be prompted with an error message that asks for a redesign of the ordering plan.

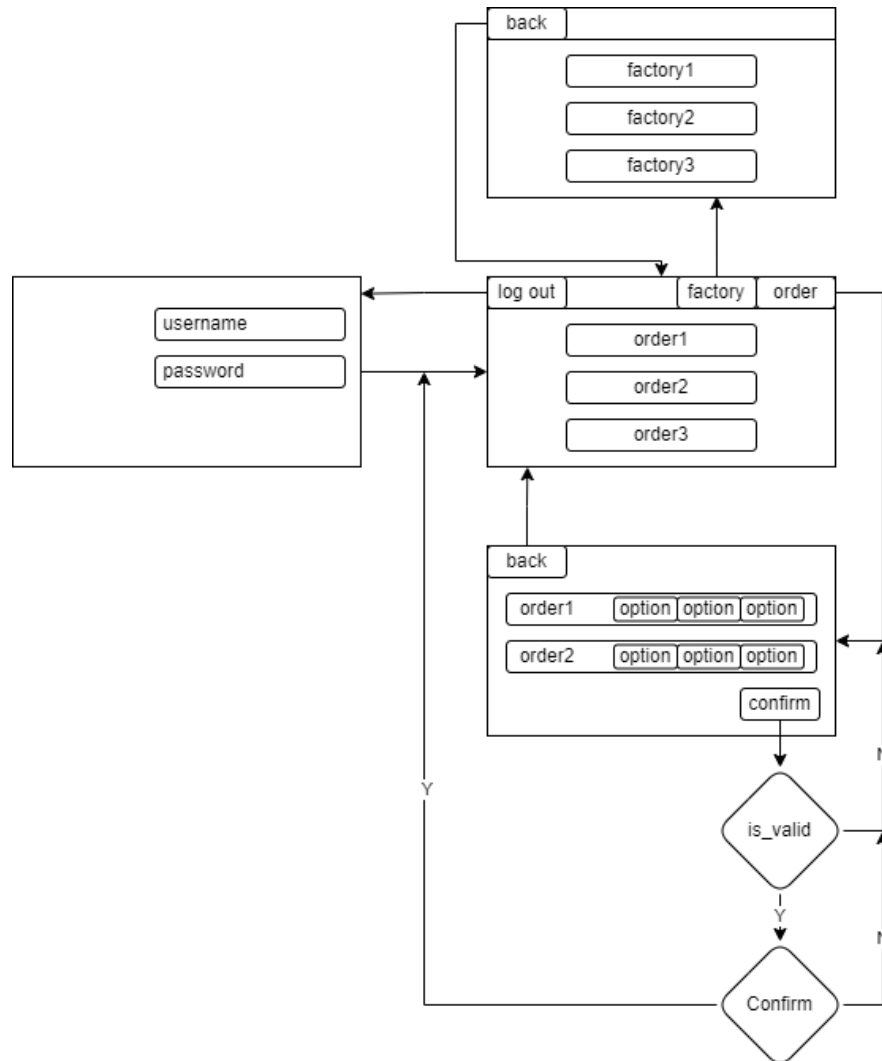


Figure 2: *Finite State Machine* Diagram

3.2 Trading System Back-End

Firstly, a simple ER diagram is designed based on the requirements and some elaboration provided by the project guideline:

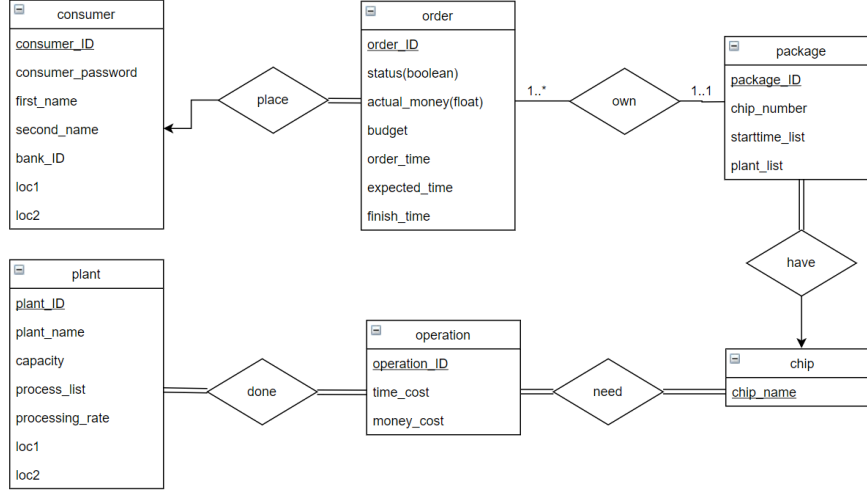


Figure 3: *ER* Diagram

We assume the customer will provide an order with different sets of chips and different volumes. Then, the system, designed by the team, will provide a score for these ordering decisions. After that, this order will be executed by the system, where relevant information will be stored in our database. Lastly, all responses and feedback will be replied back by the back-end system to the web to display all necessary information about this order.

We decide to use MySQL for this project. The next priority is to write a connector that connects MySQL and the back-end system (python based). Fortunately, MySQL-python connector can perform the job really well. So, we first wrote a simple script to create all tables provided by the ER-diagram above:

3.3 Banking System

We also designed a banking system for this project. This banking system has an independent database to store the information of each user since we wanted to highlight its independence.

The system supports the reading of a user's bank card ID and the amount

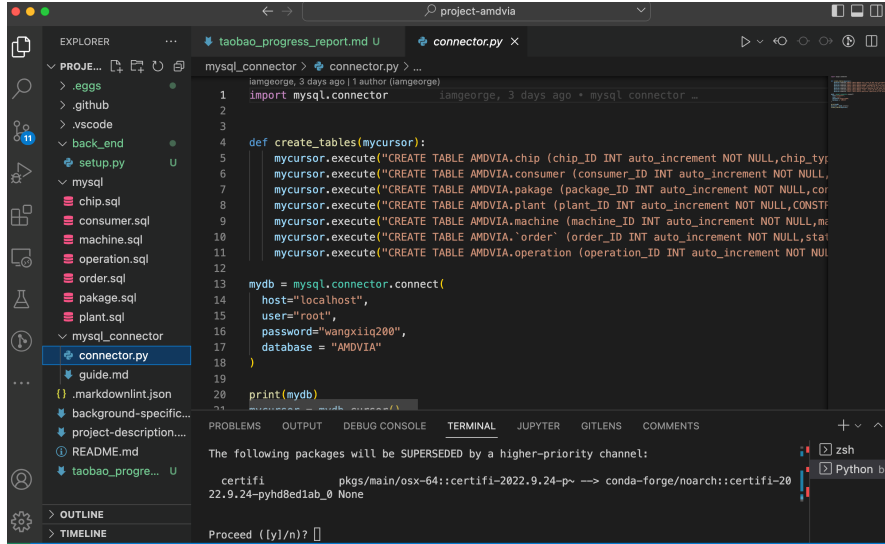


Figure 4: *Code Presentation Diagram*

to be deducted. After receiving the deduction request, the user's balance and deduction amount will be compared in the database. If the user balance is insufficient or the ID of the corresponding user cannot be queried, the message of deduction failure will be returned. The data in the database will not change. Otherwise, the message of successful deduction will be returned and the user's balance in the database will be updated.

3.4 Target Analysis

A data analysis team is set up to analyze the user's input production policy and return the score (KPI) for that policy. This team will further study the influence of each variable on the output KPI.

Our analysis is based on the input of customers and the information stored in our database. Basically, we settled down the order, which include different types of chips and their corresponding amount. Then we settled down the plants' appointment lists, which show when they are available in the future. Under these two conditions, we find the corresponding chip information and use the random

simulation and kernel density estimation to measure how good is the production plan proposed by the customer. We test our method on three different levels of occupation situations among 200 factories to show its usability.

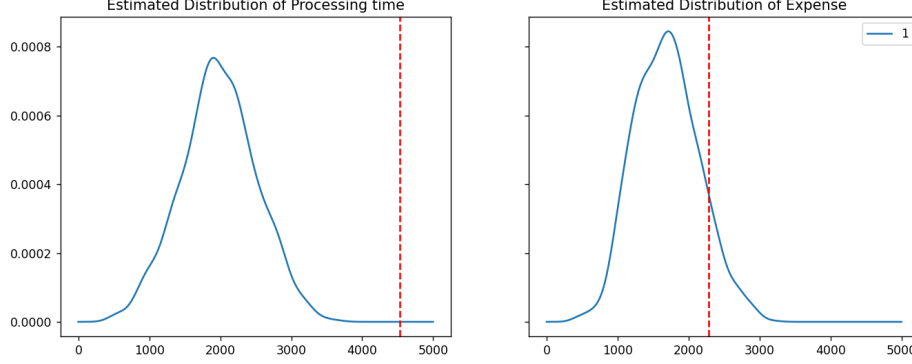


Figure 5: *Real-time Simulation Result* Diagram

This is a real-time simulation result that we will show to the customer. The left figure is the for processing time, and the right figure is the for the expense. The blue curves are the estimated density function. The red line indicates the rank of the customer's plan in expense and processing time. Then we use weights to summit the area under these two curves and get the final KPI.

4 Program Design

The overall design idea of this project can be found in the Historical Process part (page 2), so we won't go into details in this part. This part will mainly introduce the related design of the back-end database

4.1 ER Diagram

Here shows our ER diagram of the main database. We could start from the consumer table. As you can see, the consumer table includes a unique ID,

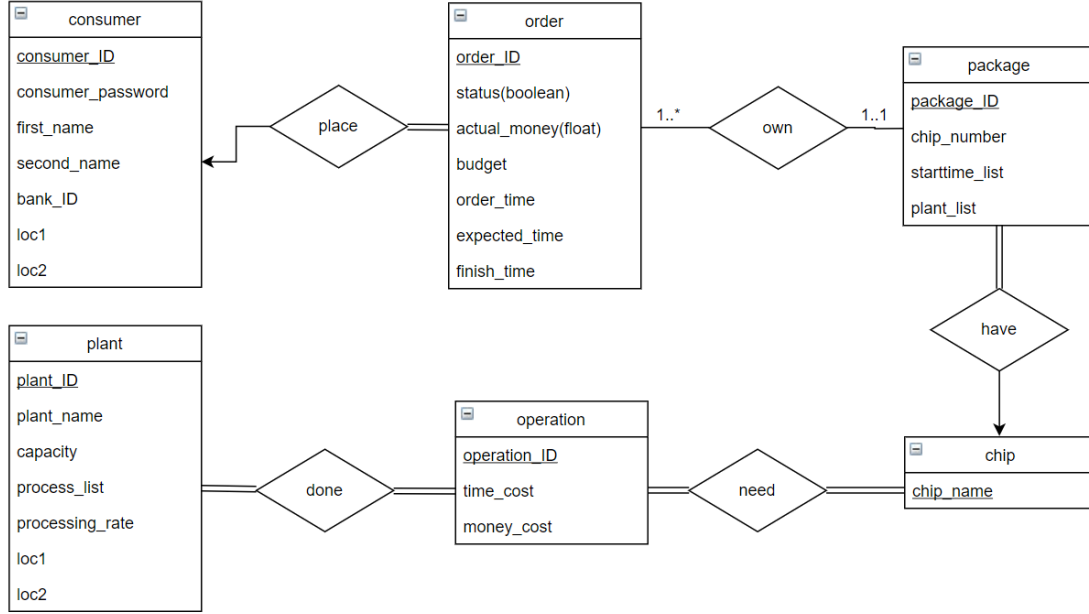


Figure 6: *ER* Diagram

and other necessary information including location, bank ID, and so on. It has a one-to-many relationship with the order table. Since each order must have a consumer, one consumer can have several orders and some consumers might not have any orders. Here one order contains many types of chips. So we need to depart orders into different packages, where one package only contains one type of chip. So one order can have one-to-many packages, and each package must belong to one order. We also use unique order and package IDs as the primary key. Then we can tell that each package must have one type of chip, and one chip needs several operations to produce it. Last but not least, operations need to be done by plants. Since one type of machine can do several types of operation, and one operation can be done by different plants, this is a many-to-many relationship.

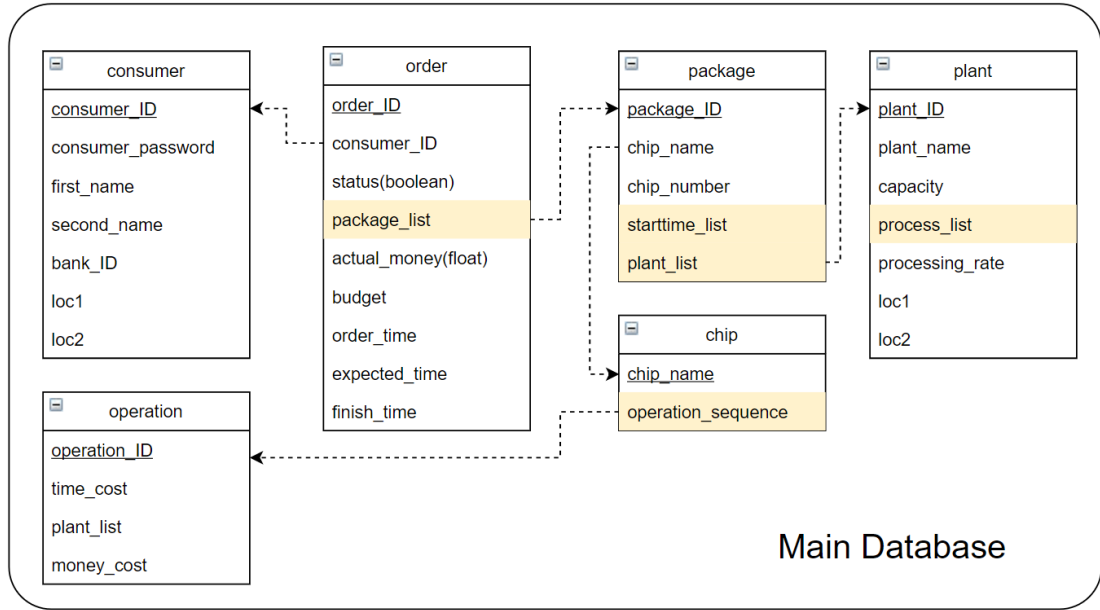


Figure 7: *Relation Schema* Diagram

4.2 Relation Schema

Then we designed the relation schema. The orange attributes are multi-valued attributes that are stored as Jason. The dash lines indicate the original foreign keys. Actually, we don't use tables to represent the relationship in the ER diagram. Instead, we use these lists, for example, process lists, and package lists to store them as an attribute without normalization. Also, the reason why we use a dash line to indicate the foreign keys is that we don't use any foreign keys in implementation.

Firstly, for no normalization, the reason is that in the analysis of the production plan, we need to extract this information frequently. If we depart the relationship into several tables, we need to do many natural joins when we extract the information, which is a waste of time. Instead, storing them as multi-valued attributes results in easier and faster extraction in analysis.

Besides, we don't use any foreign key constraint because our database

doesn't allow any input that violates the constraints. More specifically, users are only allowed to select the legal information that is already settled in our system at the front end, the referenced attribute will be determined all by our system instead of the customer themselves. This means violation of foreign key constraints will not happen! This eases our database and increases the processing time in the analysis. Also, since the referencing attributes such as operations of each type of machine are stored as a list in Jason, it's hard for MySQL to check the foreign key constraints.

5 Functionality Implementation

5.1 Front-End Implementation

we implemented the following front-end pages using Unity to provide service to the user. The interaction logic is written in Cand communicate with the back-end module through JSON format messages which we will late explain in detail.

1. login page

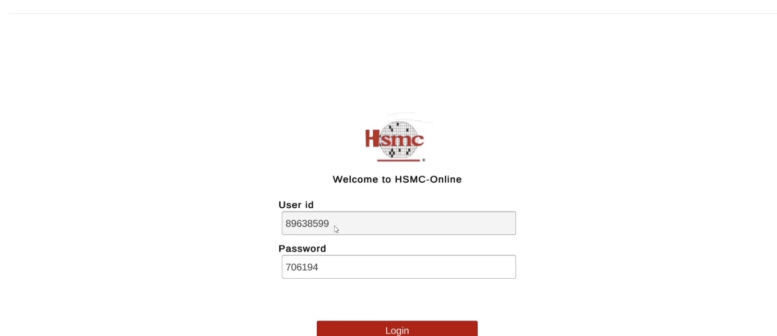


Figure 8: *Login* Page

The login page provide the user with a entrance to login to our website.

2. Order list page

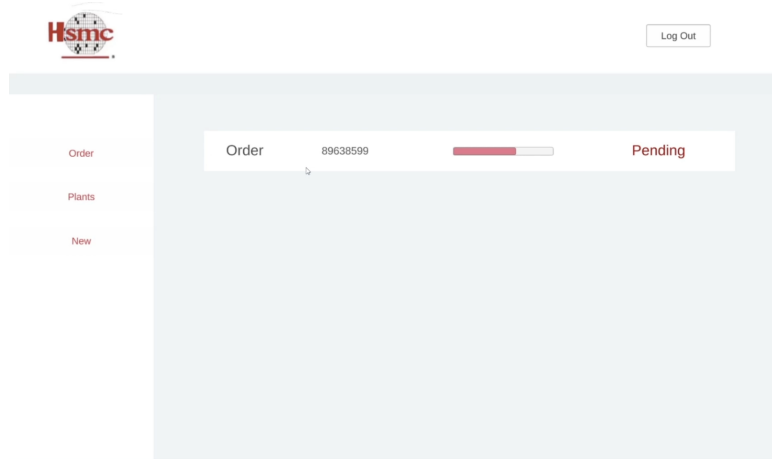


Figure 9: *Order List* Page

The order list page shows all the orders that the user made in the past.

Displays the progress and status.

3. Factory info page

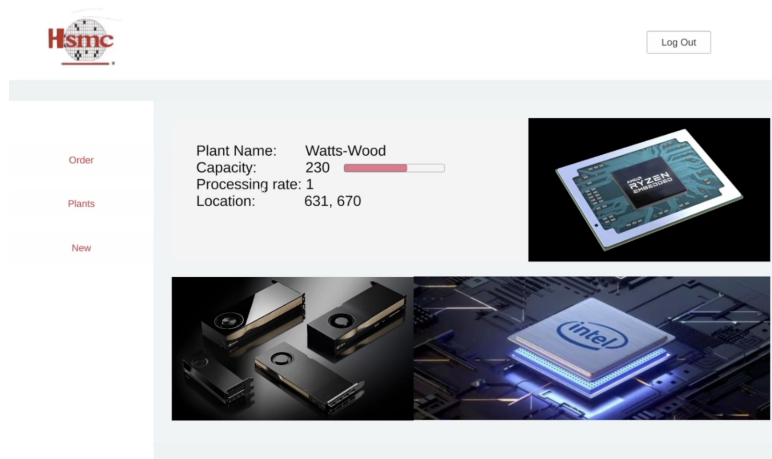


Figure 10: *Plant Info* Page

The factory info page provide the user with detailed information of each factory.

4. Order making page

The screenshot shows the 'Order Submit Page' of the Hsmc system. On the left is a sidebar with 'Order', 'Plants', and 'New' links. The main area contains a form for submitting an order. It includes a date field set to 07/01/2023, a dropdown menu showing 'NB666', and a 'Pick Process Time' button. Below these is a calendar for December 2022. To the right of the calendar are four dropdown menus for selecting different entities: 'Douglas-Franklin', 'Brown Inc.', 'Mooney-Taylor', and 'Reed, Spence and Harris'. At the bottom right of the form is a red 'Confirm' button. A 'Log Out' button is located in the top right corner of the page.

Figure 11: *Order Submit Page*

The order making page allow user to create, edit and submit a new order to the platform and receive feedback.

This screenshot shows the 'Analysis Prompt Page' with a 'Congratulations!' modal window. The modal informs the user that their plan is valid and outperforms 84.73% of other plans. It displays the 'Cost: 5622.29' and 'Time: 99'. Two line graphs are shown, each titled 'Histogram Distribution of Parameters', comparing the user's plan to others. The background shows the same order form as in Figure 11, with the 'Confirm' button visible at the bottom right.

Figure 12: *Analysis Prompt Page*

5.2 Back-End Implementation

We implement the back-end according to the process. The general process is as follows: The user submits the plan; the plan is approved and kpi is calculated

(/kpi); after the user confirms, try to debit his bank account (/pay/); after the deduction is successful, write the order information into the database (/confirm).

1. User login in the front end:

Give consumer_id, consumer_password; the back end returns password error (0), id does not exist (2), successful login (1).

- Interface: [post] http://10.31.133.149:5000/checklogin

- Correct example:

```
1      {
2          // The json sent by the front end
3          "consumer_id": 89638599,
4          "consumer_password":706194
5          // Output is 1
6      }
```

- Wrong example:

```
1      {
2          // The json sent by the front end
3          "consumer_id":89638599,
4          "consumer_password":400820
5          // Output is 0
6      }
```

- Example of username does not exist:

```
1      {
2          # The json sent by the front end
```

```

3         "consumer_id": 11111111,
4         "consumer_password":706194
5     # Output is 2
6 }

```

2. The user switches to the order query page, and the back-end returns the status of the order.

- Interface: [get]http://10.31.133.149:5000/orderrecordconsumerid=89638599
- Output example:

```

1     [(12345678, 89638599, 0, '{"0": 12, "1": 84}', 562229,
      60005, 2, 97, 99)]
2     # tuple listervervey tuple is an order
3     # tuple: (order_id, consumer_id, status, package_list,
      actual_money, budget, order_time, expected_time,
      finish_time)

```

3. The user switches to the factory information page, and the back-end returns the information status of all current factories.

- Interface: [get]http://10.31.133.149:5000/plant/process
- Output example:

```

1     {0: [], 1: [], 2: [], 3: [[55, 64], [80, 89]], 4: [],
      5: [], 6: [[0, 20], [20, 40]], 7: [[25, 47], [47,
      69]]}
2     # The actual quantity is more

```

4. After the user switches to the ordering interface and selects a chipname, the back-end returns information about the chip, including the number of operations and which plants each operation can be used for.

- Interface: `[get]http://10.31.133.149:5000/chip/process?chiptype=DM878`

- Order info example:

```
1      # The json returned by the back-end
2      {
3          "operation1": [
4              "Brown Inc",
5              "Davis PLC",
6              "Hendricks, Price and Lewis",
7              "Miller PLC",
8              "Woods Inc"
9          ],
10         "operation2": [
11             "Lowery Ltd",
12             "Lowe-Mcintyre",
13             "Duncan Ltd",
14             "Bridges-Hernandez",
15             "Solis-Spencer",
16             "Jones Ltd",
17             "Watts PLC",
18             "Powell Group"
19         ],
20         "operation3": [
```



```

21         "Blankenship, Howell and Howell",
22
23         "Fisher, Warren and Moore",
24
25         "Young, Taylor and Jones",
26
27         "Fox Ltd",
28
29         "Ruiz-Jones",
30
31         "Bates, Nelson and Sanchez"
32     ],
33
34     "operation4": [
35
36         "Mooney-Taylor",
37
38         "Alvarado PLC",
39
40         "Mckee-Esparza",
41
42         "Robinson LLC",
43
44         "Davis, Murphy and Osborne",
45
46         "George LLC"
47     ],
48
49     "operation5": [
50
51         "Reed, Spence and Harris",
52
53         "King-Oneal",
54
55         "Arnold-Carlson",
56
57         "Reed, Johnson and Mitchell",
58
59         "Hall, Woods and Sanchez",
60
61         "Douglas-Franklin",
62
63         "Stone Group",
64
65         "Lewis Inc",
66
67         "Pace, Powell and Stone",

```

```

46         "Clark-Ford"
47     ],
48     "operation_number": 5
49 }

```

5. After the user confirms the order for the first time, the back-end checks whether the user's strategy is reasonable, returns the unreasonable reason if it is unreasonable, and analyzes and returns KPI if it is reasonable. After it is reasonable, two pictures (money.jpg and time.jpg) will be generated in the bank end folder to represent the distribution under random conditions

- By default, the factory selected by the user can perform the corresponding operation, and this part only checks the rationality of starttime.
- Interface: [post] <http://10.31.133.149:5000/kpi>
- Success example:

```

1         # If the user's policy is reasonable, a dict is
           returned
2
3         # input
4         # The json sent by the front end
5         {
6             "status":"analysis",
7             # status must be analysis otherwise no analysis will
               be done
8             "order_id":12345678,

```

```

9         "consumer_id":89638599,
10        "package_id_list":[12,84],
11        "budget":60005,
12        "expected_time":97,
13
14        "0_chip_name":"CK101",
15        "0_chip_number":50,
16        "0_plant_name_list":["Reed, Spence and Harris","Diaz,
17                               Andersen and Cooper","Fernandez, Boyd and Palmer"],
18
19        "0_starttime_list":[0, 25, 55],
20
21        "1_chip_name":"CK101",
22        "1_chip_number":30,
23        "1_plant_name_list":["Reed, Spence and Harris","Diaz,
24                               Andersen and Cooper","Fernandez, Boyd and Palmer"],
25
26        "1_starttime_list":[20, 47, 80]
27    }
28
29    # output
30    {
31        "finishtime_list": [[20,47,64],[40,69,89]],
32        "kpi": 84.73384896174014,
33        "money_cost": 5622.29,
34        "time_cost": 99,
35        "validate": true

```

- Defeat example:

```
1      # If fail, a string is returned to indicate the
      unreasonable reason.

2

3      # input

4      {

5          "status": "analysis",

6          "order_id": 12345678,

7          "consumer_id": 89638599,

8          "package_id_list": [12, 84],

9          "budget": 60005,

10         "expected_time": 97,

11

12         "0_chip_name": "CK101",

13         "0_chip_number": 50,

14         "0_plant_name_list": ["Reed, Spence and
                                Harris", "Diaz, Andersen and Cooper", "Fernandez,
                                Boyd and Palmer"],

15         "0_starttime_list": [0, 25, 55],

16

17         "1_chip_name": "CK101",

18         "1_chip_number": 30,

19         "1_plant_name_list": ["Reed, Spence and
                                Harris", "Diaz, Andersen and Cooper", "Fernandez,
```

```

        Boyd and Palmer"],
20      "1_starttime_list": [20, 43, 80] #Here the second
        number changed to 43
21
22      #output
23      #the 2-th start time in package 2 in plant "Diaz,
        Andersen and Cooper" is occupied
24    }

```

6. Attempt to charge user

- Interface: [post]http://10.31.133.149:5000/bank/pay
- Success example:

```

1      # input
2      {
3          "id" : 89638599,
4          "money" : 1234 //
5      }
6
7      # output successful

```

- Fail example:

```

1      # input
2      {
3          "id" : 89638599,
4          "money" : 1234123123

```

```

5         }
6
7         # output
8         # 503 service unavailable: The account doesn't have
          enough money!

```

7. After the bank debits, store the order data in the database of order and package (it must be done after the success of steps 5 and 6)

- Data example:

```

1         # input
2         {
3             "status": "confirm",
4             "order_id": 12345678,
5             "consumer_id": 89638599,
6             "package_id_list": [12, 84],
7             "budget": 60005,
8             "expected_time": 97,
9             "money_cost": 5622.29,
10            "time_cost": 99,
11            "order_time": 2,
12            "finishtime_list": [[20, 47, 64], [40, 69, 89]],
13
14            "O_chip_name": "CK101",
15            "O_chip_number": 50,
16            "O_plant_name_list": ["Reed, Spence and Harris", "Diaz,

```

```

        Andersen and Cooper","Fernandez, Boyd and Palmer"],
17     "0_starttime_list":[0, 25, 55],
18
19     "1_chip_name":"CK101",
20     "1_chip_number":30,
21     "1_plant_name_list":["Reed, Spence and Harris","Diaz,
        Andersen and Cooper","Fernandez, Boyd and Palmer"],
22     "1_starttime_list":[20, 47, 80]
23 }
24
25     # output successful

```

5.3 Data Analysis Implementation

For the analysis, we need to first introduce some basic concepts. Given the settled plants' status and order, we have multiple ways to produce them. So our KPI measures how good is your production plan among all possible plans and shown as a score. We basically consider two aspects to measure the KPI: expense and production time. These two aspects require us to consider both the production of each plant and the transportation of different plants.

This program *Analysis.py* stores our analysis algorithm.

We first extract the data we need, which is specified on the last page, then we perform a simulation algorithm to generate different legal production plans under the same situation as the customer's plan. We then believe that the expense and the processing time of these simulation plans will approximate

the true distributions respectively if our simulation plans are unbiased. Then the expense and the processing time of the customer's plan indicate its rank among all these simulation plans. We then use weights to summit the ranks of expense and processing time and get the final KPI.

We used random simulation and keep it seems to be unbiased. The algorithm is: we randomly ordered the chips in the package and assigned the factory to the chips. Because the order of assigning chips may influence the processing time and the expense, then for each type of chip, we follow the manufacture sequence, i.e., operation1, operation2... , to assign a factory for each operation. For operation i , we first randomly pick a factory from all the factories that can implement operation i . We then calculate the processing time and the expense that this factory needed to do operation i (including the transition time and expense from the last factory). After that, we check the status of the picked factory and find enough spare period for it to implement operation i . We will then update the status of the factory (with the label 'simulation' to indicate that this arrangement is not real). Then we randomly generated a small time gap to simulate the unwise choice and waste of time, which may happen in real plans. Then we repeat this procedure so on so forth. After one round of random assignment of an order, we will get a decision and its corresponding processing time and expense. By repeating this procedure a thousand times, we can approximate the distribution of the processing time and the expense corresponding to a specific package.

Then we utilize Kernel Density Estimation with Gaussian kernel to estimate the approximate distribution and transform it into a continuous distribution. The left figure is the KDE for processing time, right figure is the KDE for the expense.

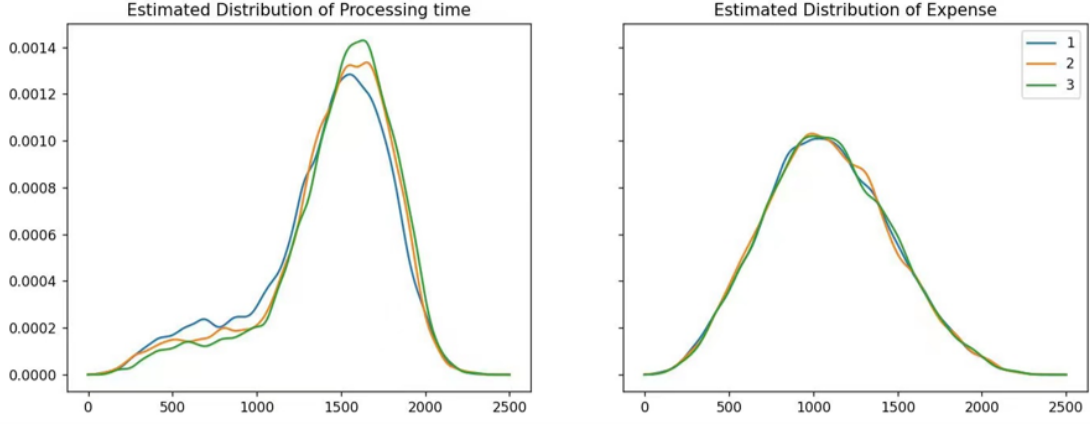


Figure 13: *KDE Diagram*

We test our method on three different levels of occupation situations among 200 factories. And the result is reasonable since the blue curve is tested on the slightest occupation situation, and the mean processing time is obviously smaller, but the expense is similar to other situations.

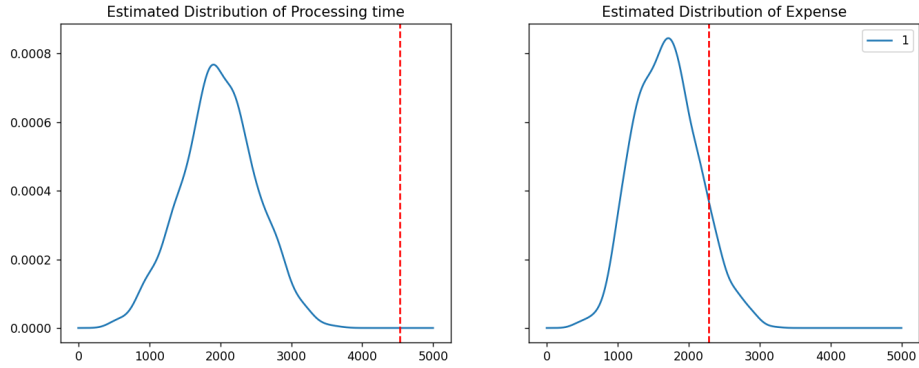


Figure 14: *Real-time Simulation Result Diagram*

This is a real-time simulation result that we will show to the customer. The blue curves are the estimated density function. The red line indicate the rank of the customer's plan in expense and processing time. Then we use weights to summit the area under these two curve and get the final KPI.

6 Conclusion

In conclusion, we designed a database for a chip-order website. We have designed and implemented a simple and beautiful web interface, which is convenient for customers to purchase chips and view order records. We also designed the ER diagram and reduced it to a normalized relational schema. We have made some fake data for the preliminary test and improved the authenticity of the data by consulting relevant information in the later stage. Using the statements of python, automatic updates of the database are realized. We also wrote an analysis program to visualize the degree of the feasibility of the order, which can provide reference for customers.

7 Contribution

<i>Task</i>	<i>Member (in order of student ID)</i>
Architecture design	All
Front-end	Huang Xizheng, Lin Zexin, Zhou Yuxiao, Sun Shaoqiang
Back-end	Wang Xi, Peng Qiaoyu, Deng Yixuan, Qu Hengyi, Xiong Yiming
Data analysis	Peng Qiaoyu, Qu Hengyi, Xiong Yiming, Liu Yuxuan
Report & Pre	All

— End of Project Report —