TymeX Internship Program 2024

AI Intern Report

Nguyen Nhat Quang

November 2024

1 E-Commerce Response System Design

You are working for an e-commerce company. Design a system that can respond to any user query regarding the shipping status of their order.

1.1 System Requirements

The system should be abled to:

- Work correctly (return information on order status, estimated delivery time and shipping progress)
- Identify and verify users through their order IDs and customer IDs.
- Secure from methods like SQL injections and other stuffs.
- Access real-time shipping information from databases (this is depends on the e-commerce company)
  - Some e-commerce companies have their own shipping department like Shoppe, Lazada.
  - Some e-commerce companies/platform doesn't have their own shipping department like TikTok shop (they outsourcing this part by T&T).
- Have users friendly UX/UI.
- Flexible since the customers may forgot their order IDs or some customers will buy lots of things, which leading to a customer may have many orders at once. Flexible mean that the customer can type in the customer ID and see all the shipping things and their status.

1.2 High level system design

The system should include:

- Having API gateways for users to query the order status.
- Order management service: process and store order details, store things like shipping status, shipper, the person who reponsibles for that stage.

- Database: using relation databases like MSSQL to store order details and Key-value store like Redis to get frequently accessed information.
- Having friendly UX/UI for users to interact with.
- Having a system to send real-time updates to users when there is a change in the shipping status of their orders

1.3 Data Flow

1.3.1 User Query Flow

1. User sends a query request through the API Gateway.
2. Order Management Service checks Redis cache for the latest status; if not present, it queries the main database.
3. Response is sent back to the user with the current order status and tracking information.

1.3.2 Status Update Flow

1. External Integration Service fetches updates from third-party logistics or the company's shipping department itself and sends them to the Tracking Update Service.
2. Updates are written to the database and cache, and a notification is triggered if required.

1.4 Security Considerations

- Having secure communication to ensure TLS encryption and user authentication (e.g., OAuth) at the API Gateway.
- Having third-party security to ensure that access to third-party logistics APIs is secured with API keys or OAuth.

2 Prototype System with Scheduling for Multiple Users

2.1 Objective

Design a prototype system that efficiently manages multiple user requests, providing each user with accurate shipping information while handling concurrent queries.

2.2 Assumptions

Some assumptions about how this system works is as following:

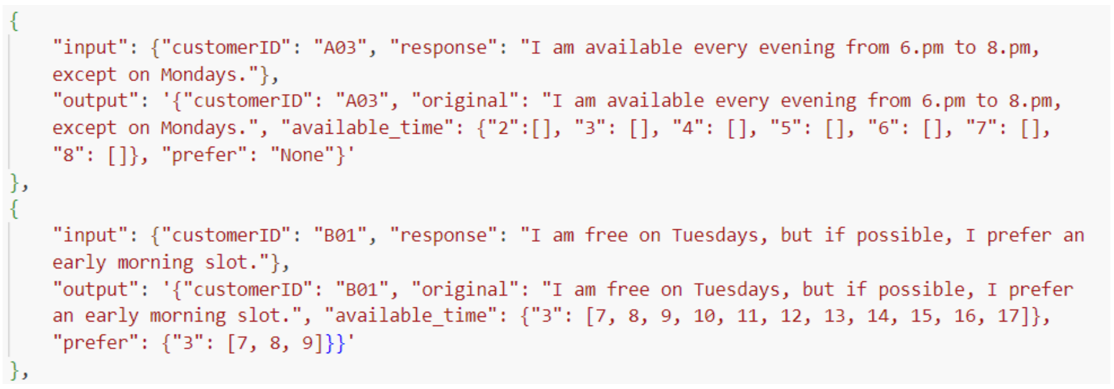
- All meeting hours are correctly at that hour, somethings like 7:00, 8:00, 9:00 instead of 7:12, 8:12, 9:30.
- There should only be 50 customers engage in the system.
- All this setup is for next week, to avoid something like if today is friday and the customer say that he/she is free on monday, it will be understanded that they are talking about scheduling for next week. Therefore as long as they submit their responses at the end of week the algorithm will no prioritize anyone.
- Customer can set up schedule from Monday to Sunday (in the code i denoted Monday is 2 and Sunday is 8), from 7AM to 5PM (in the code i denoted 7AM is 7 and 5PM is 17 (i am using 24hour scale)).
- Each customer can reassign again, which is in case something surprised happened, they can change their schedule.
- Each customer can have favorable days or hours over others, therefore the weight will be adjust higher (discussed more careful in algorithm design).

2.3 Scheduling Algorithm

The scheduling algorithm for this prototype include 2 steps.

1. **Convert text to json predefined format**.

In this step, the user will answer their responses on the system, make an API call to the server, the server will then use a LLM to convert that response into another JSON response (However this time, the content is different). It will be something like this:



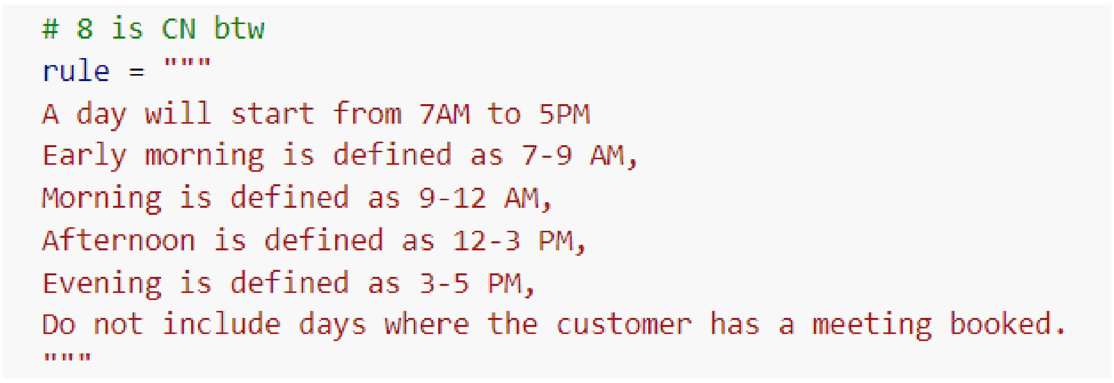
```
{
    "input": {"customerID": "A03", "response": "I am available every evening from 6.pm to 8.pm,
    except on Mondays."},
    "output": '{"customerID": "A03", "original": "I am available every evening from 6.pm to 8.pm,
    except on Mondays.", "available_time": {"2":[], "3": [], "4": [], "5": [], "6": [], "7": [],
    "8": []}, "prefer": "None"}'
},
{
    "input": {"customerID": "B01", "response": "I am free on Tuesdays, but if possible, I prefer an
    early morning slot."},
    "output": '{"customerID": "B01", "original": "I am free on Tuesdays, but if possible, I prefer
    an early morning slot.", "available_time": {"3": [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]},
    "prefer": {"3": [7, 8, 9]}}'
},
```

Figure 1: LLM output

The reason for this is that the customer will type in a prompt, normal approach like rule based to extract cannot be efficient because language is wide. Therefore i decide to use a LLM approach to extract these information better. This is what the output look like (remember that it's a string)

```
{
  "customerID": "Unique identifier for the customer",
  "original": "Original text describing customer's
              availability and preference",
  "available_time": {
    "day_of_week": [List of available hours
                    for the customer on that specific day]
  },
  "prefer": {
    "day_of_week": [List of preferred hours
                    for the customer on that specific day]
  }
}
```

Another problem is when some customer will say "I prefer early morning". However early morning is what so vague, therefore in the prompting process, i specify some rules like this:



```
# 8 is CN btw
rule = """
A day will start from 7AM to 5PM
Early morning is defined as 7-9 AM,
Morning is defined as 9-12 AM,
Afternoon is defined as 12-3 PM,
Evening is defined as 3-5 PM,
Do not include days where the customer has a meeting booked.
"""
```

Figure 2: Rules for output

The LLM will then convert these terms or very similar terms such as "i will be appreciate if we can meet in the very morning" into dictionaries of times in week.

However guiding LLM to output JSON string like this is very difficult, eventhough there are some libraries, packages and function have been conducted to express this problem, it still not working like what i have expected. I do try to finetuning some models and perform few-shot learnings, but the results are not as good as what i expected. Therefor in the next steps, i will assume that the LLM has already return the output beautifully (sorry but i am finding a way to solve this problem).

2. **Scheduling algorithm**

In this step, i assume that the LLM has already extract the json-string format correctly and beautifully (however it's not, i am fixing it, but i affraid that i can not do it on time).

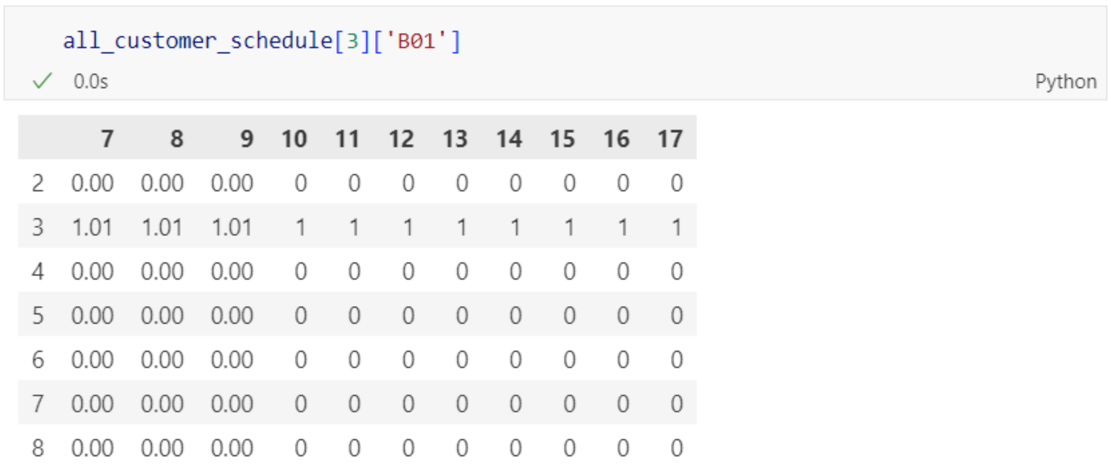
Now for each user, we will have a json file like this:

```
{
    "customerID": "A01",
    "original": "I am available every morning
                        from 9.am to 11.am, except on Wednesdays.",
    "available_time": {
        "2": [9, 10, 11],
        "3": [9, 10, 11],
        "4": [],
        "5": [9, 10, 11],
        "6": [9, 10, 11],
        "7": [9, 10, 11],
        "8": [9, 10, 11]
    },
    "prefer": "None"
}
```

Since the model failed, i will reuse the output that i written by hand. I assume that these results are what the LLM produces, therefore it is a string, but look like json format. Therefore i will convert these strings back to dictionaries and put it into a large list.

The next step is the important step, i will create a function to create dataframe that represents the availability of each customer, this dataframe will have the shape of (7×10) because there are 7 days in week and i assume that the company will have meeting only from 7AM to 5PM. If a customer is available at some specific times in day, the value of the index (days,times) will be set to 1 and for those customers who prefer some specific times and days, i will add 0.01 to each customer time if they prefer it. Take the customer with customerID=B03 for example, we can see in figure 1 that this customer is free on tuesday and prefer early morning. Therefore the values at those time on Tuesdays will have a value of 1.01, other time on Tuesday will be 1, and other time other than Tuesday will be 0 (because he is not free) just like in figure 3:

We then apply this function for all customer and stored it in a list, finally we will have a list with n elements where each element is a availability dataframe. After that we use the values of these dataframe to obtain the



```
all_customer_schedule[3]['B01']
```

```
✓ 0.0s                                                    Python
```

	7	8	9	10	11	12	13	14	15	16	17
2	0.00	0.00	0.00	0	0	0	0	0	0	0	0
3	1.01	1.01	1.01	1	1	1	1	1	1	1	1
4	0.00	0.00	0.00	0	0	0	0	0	0	0	0
5	0.00	0.00	0.00	0	0	0	0	0	0	0	0
6	0.00	0.00	0.00	0	0	0	0	0	0	0	0
7	0.00	0.00	0.00	0	0	0	0	0	0	0	0
8	0.00	0.00	0.00	0	0	0	0	0	0	0	0

Figure 3: Availability dataframe of B01

final dataframe, now each element in this dataframe is a value ranging from 0 (no customer free at that time) to $1.01 \times n$ where n is the number of customer (this is when all customers are free and enjoy having a meeting at that time). In order to make sure that we want all customers to engage in the meeting, we can also set up a minimal value equal to the number of customer. This minimal value ensure that we won't missing any times in week that all customers are free and can engage in the meeting.

The reason for preferred score to be 0.01 is to avoid a scenario where if we set this score to be 1 or more than one, since a large amount of customers came in to the meeting, the algorithm will choose some time slots that not having all the customer. For example if there are 5 customers, 4 of thems are free on everyday but prefer to meeting in the evening, meanwhile the 5th customer is only free in the morning. In that case, if we set the prefered score to be 1, the total score will be 8 (2×4) and which is much higher than 5 (5 customers), the algorithm will then choose evening where only 4 people can attend and left the solution meeting in the morning so that 5 customers can engage.

2.4 Conclusion and further approach

The hardest part of this problem is somehow to make the LLM to output the format beautifully and correct, it should some how be able to capture the meaning of the customer and correctly follows the rule to convert the input to the correct output.

The scheduling algorithm in this case work quite well. It can extend to more customers if we just lower the preferred value. However this approach can not capture minutes like 9:27AM or 3:21PM and stuffs, therefore we will need to readjust the way we store the availability of each customer to better suit the situation.