Yuan Cheng
2022.4.19

Final Project Data Checkpoint

In the final project, I created a project which can help the users to find the restaurants nearby according to the users' input criteria.

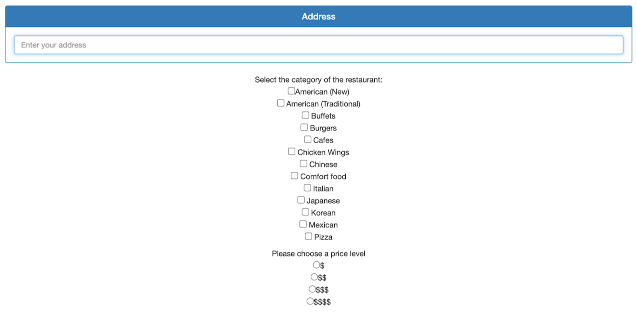
Project Code:

https://github.com/YuanChengRua/SI507-Final-Project

Data Sources

Used data sources so far:

- **Yelp Fusion businesses search Endpoint**: https://www.yelp.com/developers/documentation/v3/business_search
- Formats: JSON
- Data Access and Caching:
    - Register an API key on Yelp Fusion to enable the developers to access the data
    - Design an HTML form page to collect users' preferences



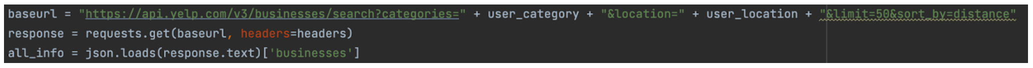
Address

Enter your address

Select the category of the restaurant:
☐American (New)
☐ American (Traditional)
☐ Buffets
☐ Burgers
☐ Cafes
☐ Chicken Wings
☐ Chinese
☐ Comfort food
☐ Italian
☐ Japanese
☐ Korean
☐ Mexican
☐ Pizza

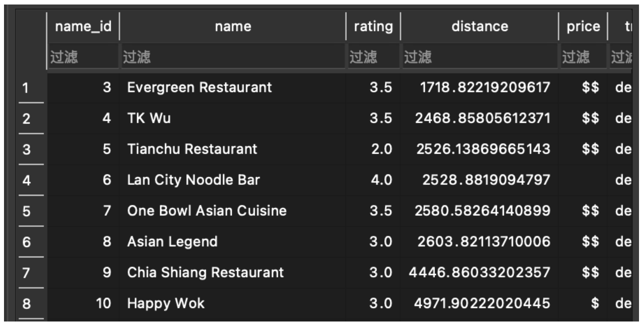
Please choose a price level
○$
○$$
○$$$
○$$$$

    - Apply the collected data to generate the API endpoints to access the data



```
baseurl = "https://api.yelp.com/v3/businesses/search?categories=" + user_category + "&location=" + user_location + "&limit=50&sort_by=distance"
response = requests.get(baseurl, headers=headers)
all_info = json.loads(response.text)['businesses']
```

    - The database is used to cache the data for further analysis

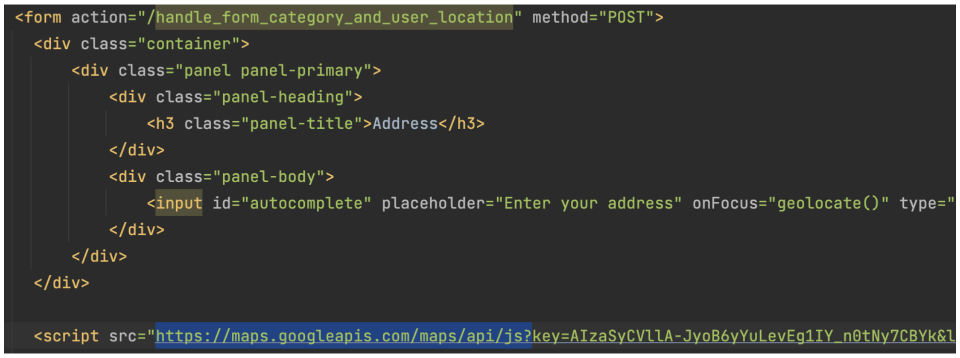


	name_id	name	rating	distance	price	ti
	过滤	过滤	过滤	过滤	过滤	过滤
1	3	Evergreen Restaurant	3.5	1718.82219209617	$$	de
2	4	TK Wu	3.5	2468.85805612371	$$	de
3	5	Tianchu Restaurant	2.0	2526.13869665143	$$	de
4	6	Lan City Noodle Bar	4.0	2528.8819094797		de
5	7	One Bowl Asian Cuisine	3.5	2580.58264140899	$$	de
6	8	Asian Legend	3.0	2603.82113710006	$$	de
7	9	Chia Shiang Restaurant	3.0	4446.86033202357	$$	de
8	10	Happy Wok	3.0	4971.90222020445	$	de

- Summary of data: Each search could result in 50 records max due to the limitation of the Yelp Fusion API design and each data contains several dimensions of a restaurant including name, rating, price, categories, image_url, restaurant URL, etc.
    - Name: represents the name of the restaurant stored in the Yelp Fusion database and we use it to identify the restaurant

    - Rating: represents the rating score for each restaurant
    - Price: The average price for a restaurant
    - Categories: Type of restaurant and each restaurant could have several different categories
    - Restaurant URL: Used to access the detail of a restaurant
    - Image_url: Used for presentation purpose while using.

- **Google MAPS API**: https://maps.googleapis.com/maps/api/js
- Data Access:
    - Register the Google Maps Static API
    - Embed the API in the HTML file to enable the auto-complement of the address



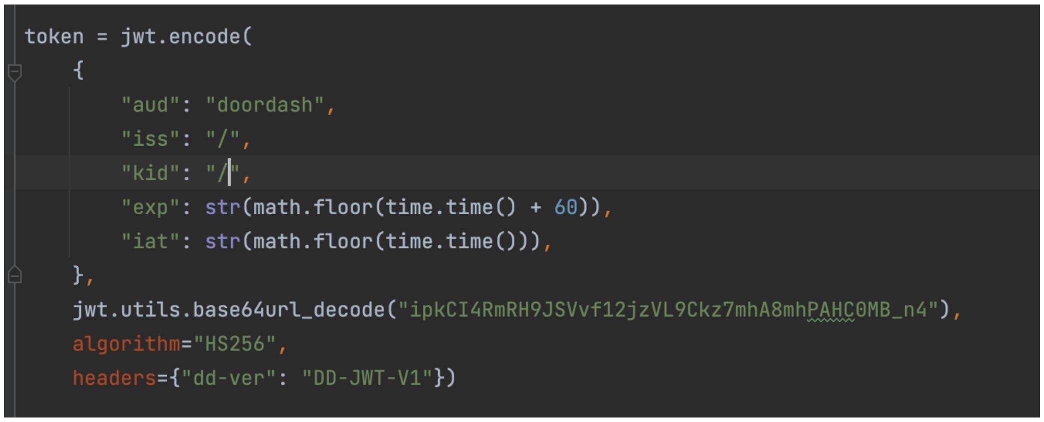
```
<form action="/handle_form_category_and_user_location" method="POST">
  <div class="container">
      <div class="panel panel-primary">
          <div class="panel-heading">
              <h3 class="panel-title">Address</h3>
          </div>
          <div class="panel-body">
              <input id="autocomplete" placeholder="Enter your address" onFocus="geolocate()" type="
          </div>
      </div>
  </div>

  <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCVllA-JyoB6yYuLevEg1IY_n0tNy7CBYk&l
```

    - No caching is used for this API

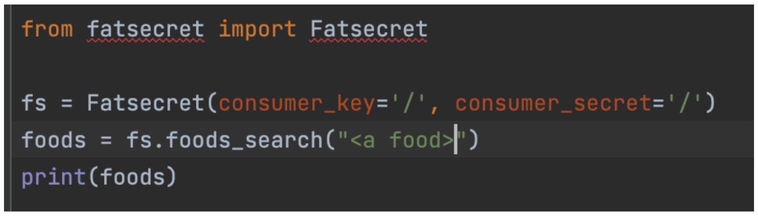
Future data source:

- **Doordash**: https://openai.doordash.com/drive/v2/deliveries
  Then token for the use of this API is designed as follows:



```
token = jwt.encode(
    {
        "aud": "doordash",
        "iss": "/",
        "kid": "/",
        "exp": str(math.floor(time.time() + 60)),
        "iat": str(math.floor(time.time())),
    },
    jwt.utils.base64url_decode("ipkCI4RmRH9JSVvf12jzVL9Ckz7mhA8mhPAHC0MB_n4"),
    algorithm="HS256",
    headers={"dd-ver": "DD-JWT-V1"})
```

  Once the user finishes selecting the food, the system will forward it to the Doordash API if the users want to deliver the food, and then, the Doordash API will simulate a delivery.
- **FatSecret Platform**:



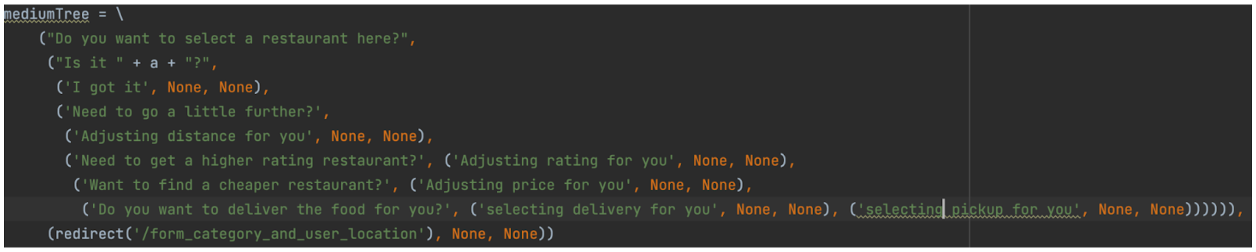
```
from fatsecret import Fatsecret

fs = Fatsecret(consumer_key='/', consumer_secret='/')
foods = fs.foods_search("<a food>")
print(foods)
```

To help the user to check the nutrition of the food they select.

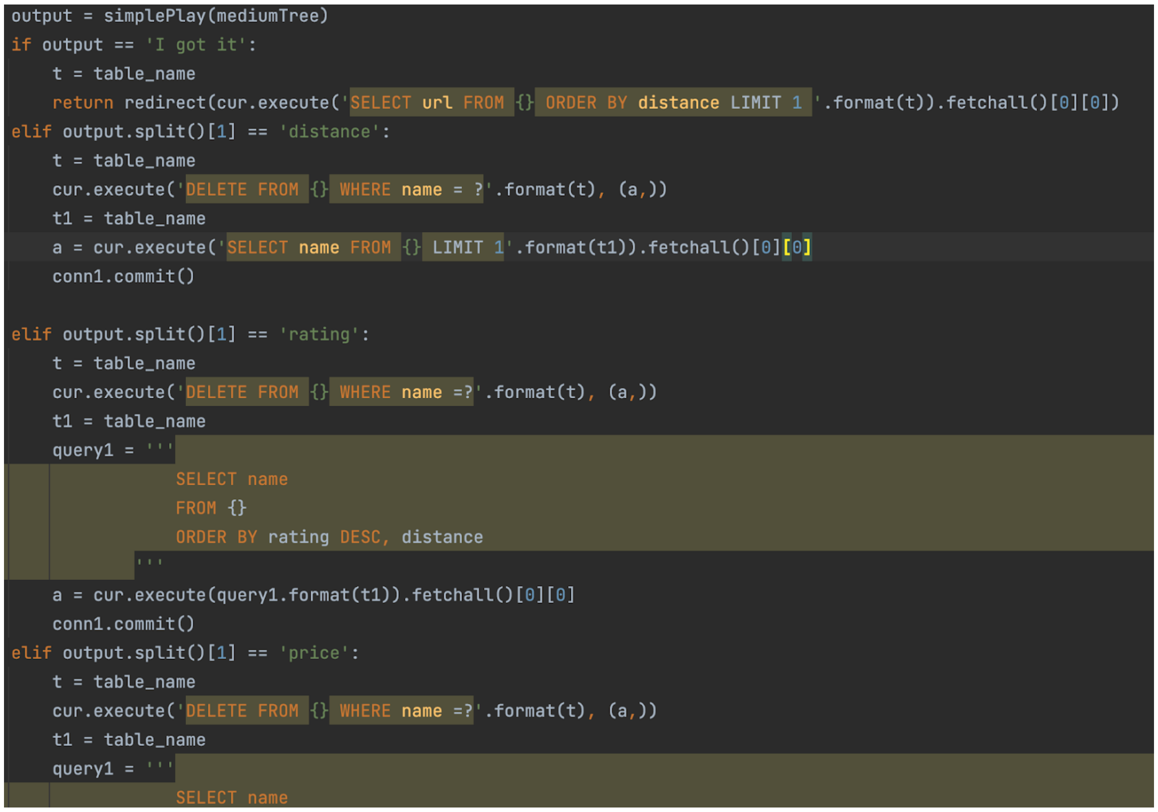
Data Structure

The data structure I have implemented is a tree structure which is similar to the structure we used in project 2 and I store the questions in a tree structure like the followings:



```
mediumTree = \
    ("Do you want to select a restaurant here?",
     ("Is it " + a + "?",
      ('I got it', None, None),
      ('Need to go a little further?',
       ('Adjusting distance for you', None, None),
       ('Need to get a higher rating restaurant?', ('Adjusting rating for you', None, None),
        ('Want to find a cheaper restaurant?', ('Adjusting price for you', None, None),
         ('Do you want to deliver the food for you?', ('selecting delivery for you', None, None), ('selecting pickup for you', None, None)))))),
     (redirect('/form_category_and_user_location'), None, None))
```

The system will initially recommend the nearest restaurant to the user and if the user like the restaurant, the system will redirect to the restaurant's page, otherwise, the system will ask the user which dimension of the restaurant needs to be changed and after the user's selection, the order of the table will be changed automatically based on the dimension selected; and the recommendation will be changed correspondingly.



```
output = simplePlay(mediumTree)
if output == 'I got it':
    t = table_name
    return redirect(cur.execute('SELECT url FROM {} ORDER BY distance LIMIT 1 '.format(t)).fetchall()[0][0])
elif output.split()[1] == 'distance':
    t = table_name
    cur.execute('DELETE FROM {} WHERE name = ?'.format(t), (a,))
    t1 = table_name
    a = cur.execute('SELECT name FROM {} LIMIT 1'.format(t1)).fetchall()[0][0]
    conn1.commit()

elif output.split()[1] == 'rating':
    t = table_name
    cur.execute('DELETE FROM {} WHERE name =?'.format(t), (a,))
    t1 = table_name
    query1 = '''
                SELECT name
                FROM {}
                ORDER BY rating DESC, distance
            '''
    a = cur.execute(query1.format(t1)).fetchall()[0][0]
    conn1.commit()
elif output.split()[1] == 'price':
    t = table_name
    cur.execute('DELETE FROM {} WHERE name =?'.format(t), (a,))
    t1 = table_name
    query1 = '''
                SELECT name
```

Each address would have a unique table in the database so every address has unique recommendations. Note that the initial data is in the order of distance and be stored in the database and when the user needs to order the data by rating, the records with the same rating will be ordered by distance and for price order, the system has the same mechanism.

When the recommendation is satisfied, the user will be redirected to the restaurant to a new page containing the details about the restaurant and the recursion will stop itself.

Interaction and Presentation

The interaction I have designed so far includes the user input criteria form and the recommendation selection process described above.
Later, I will try to implement the recommendation selection interaction on the website which makes the interaction more accessible and the presentation clear.
For more interaction, the users will be asked if they would like to get the food delivery based on their restaurant selection and they will also select the food in the restaurant and the Doordash API will do the simulation.
The future presentation includes 3 parts:

1. After the user's selection, I can use the Plotly method to display the basic information about the selected restaurant.
2. Show the user's selections and each food selected will be labeled with the nutrition facts.
3. Indicating that the delivery has been generated and showing the delivery status if necessary.

I hope the rest of the interaction and presentation will be built on the Flask framework but if the time is limited, some functionalities may not reach the expectation.