

# Food For Thought

## Final Report

---

**CIS 450/550 Fall 2018**

**Kexin Zhu, Jonathan Y Ku, Yichen Weng, Samuel Leavitt**

## **Introduction and Project goal:**

FoodForThought is a one-stop vacation planning service like no other. Currently, AirBnB is a convenient housing service that many college students use to plan their getaways during breaks. However, many people not only want a comfortable/safe place to stay, but also a convenient place. For those that believe that food is an important part of a vacation, our project is designed for you. Our project offers to provide a vacation service that allows users to find lodging surrounded by some of their favorite food preferences.

In order to give the people an idea of best Airbnb locations to stay given food preferences, our team develop a web app that combines the Yelp and Airbnb database. Specifically, we join the AirBnB New York and Yelp NYC databases to cater a Penn student's potential weekend vacation into the city. Our housing preferences allow users to decide on the neighborhood in New York he/she prefers, the price range, how many people he/she is staying with, and the type of housing (apartment, house, etc). Our food preferences include categories one would often consider. These include how expensive the restaurants are, the style of food, and how highly rated the restaurants are.

Using the user's preferences in housing and dining, we look at Airbnb housing locations suitable to the user's joint preferences. Our algorithm for finding neighborhoods suiting to a user's food preference designates a "nearby" radius around potential housing locations where we analyze nearby restaurants. We feed this query into our database, and using the results we can provide the user with information about food accommodations so as to advise him/her on the best Airbnb locations to stay given certain preferences and budget.

## **Data sources and technologies used:**

### Data sources:

1. Airbnb New York (csv file)
  - Downloaded dataset from webpage: <http://insideairbnb.com/get-the-data.html>
2. Yelp NYC (json file)
  - Fetched dataset from: <http://odds.cs.stonybrook.edu/yelpnyc-dataset/>

### Technologies:

1. Mean stack -- MongoDB, ExpressJS, AngularJS, and NodeJS
2. Handlebars and CSS for front-end displays
3. Oracle DB (to store Airbnb and Yelp databases)
4. Integration with Google Maps API
5. Use of cloud hosting (AWS & mLab)
6. Python and Pandas to clean the Yelp and Airbnb data

## **Diagram and description of system architecture:**



	comments (optional), and an array of restaurants -- each one an object with a name that acts as a foreign key referencing Yelp_data(name)
--	---

## Required Features:

### 1. Information successfully drawn from two public large datasets

We fetched the AirBnB dataset from a CSV file online through the link:

<http://insideairbnb.com/get-the-data.html>

We fetched the Yelp dataset from a JSON file through the following link:

<http://odds.cs.stonybrook.edu/yelpnyc-dataset/>

We opted to use the YelpNYC dataset instead of the larger nationwide Yelp dataset because we found that there are significant gaps in coverage where there is data for airbnb but not yelp and vice versa. For our project, we focus on New York City.

### 2. Data cleaning

We set some constraints when cleaning the data from online. When importing the data, we ran simple scripts to check for errors. First drop useless columns and check the data type for each column. Second, convert the data type if the data is in wrong data type, such as change price from varchar to number, and convert name to varchar. Finally, check the reasonability for some data, such as make sure the price range for yelp dataset is from one to four.

After cleaning all the data, convert the pandas dataframe into csv file and save local, then import all the tables into oracle.

### 3. Entity Resolution

We connected the Airbnb and Yelp data on the basis of geographical coordinates (i.e. latitude and longitude). We originally constructed a new database called "NEARBY," which contains a tuple (Airbnb ID, Yelp ID) for each airbnb-yelp pair deemed to be in close proximity (based on an arbitrary cutoff distance). This, in effect, was a cache on a join of Airbnb and Yelp data on their latitude and longitude fields.

Afterwards, we realized we could do better. We now allow the user to enter a custom limit for how far they are willing to walk from their airbnb to a restaurant. For example, if the user selects "Restaurant Search Radius: 0.37 miles," the query will join on airbnb and yelp listings that are within an approximately-square region of 0.37 miles.

Our code for joining the Airbnb and Yelp relations is given below for illustration:

```
var latdiff = parseFloat(req.params.maxdist)/69
var lngdiff = parseFloat(req.params.maxdist)/52
var jointext = "AIRBNB A JOIN AIRBNB_ADDRESS ADR ON A.ID = ADR.ID \\"
```

```

JOIN YELP_DATA Y ON ADR.LATITUDE - Y.LATITUDE > -" +
latdiff + " AND ADR.LATITUDE - Y.LATITUDE < " +
latdiff + " AND ADR.LONGITUDE - Y.LONGITUDE > -" +
lngdiff + " AND ADR.LONGITUDE - Y.LONGITUDE < " +
lngdiff;

```

Note that at New York City's geographical coordinates (40.7° N, 74.0° W), a good approximation for distance is that each degree change in latitude is roughly 69 miles and each degree change in longitude is roughly 52 miles. Note that we use a square region for now. We do this because it makes the calculations simpler and more tractable. It would undoubtedly be feasible to calculate a circular region using a more advanced algorithm such as the Haversine formula.

In the performance section, we will discuss how our method of joining Airbnb and Yelp affects performance, looking at a case where we have no optimizations, where we cache “nearby” entries using a separate database (as we did originally), and where we set up an index on both “latitude” and “longitude” for both the tables AIRBNB\_ADDRESS and YELP\_DATA.

#### *4. Normalized Relational Schema with more than four relations*

##### **AIRBNB**

(ID,NAME,LISTING\_URL,PRICE,PROPERTY\_TYPE,ROOM\_TYPE,ACCOMMODATES,NUMBER\_OF\_REVIEWS,REVIEW\_SCORES\_RATING, REVIEW\_SCORES\_LOCATION)

##### **AIRBNB\_ADDRESS**

(ID, NEIGHBOURHOOD, ZIPCODE, LATITUDE, LONGITUDE)

##### **YELP\_DATA**

(ID,NAME,URL,PHONE,LATITUDE,LONGITUDE,REVIEW\_COUNT,PRICE,RATING,TRANSACTIONS,CATEGORIES,ADDRESS,CITY,STATE,ZIP\_CODE)

##### **NEARBY**

(AIRBNBID,YELPID)

#### *5. More than one page interacting with the database*

Our project's main search page, “Start Your Journey!”, is the initial connection to the Yelp and AirBnb data, where we query our relations and return results of our search to the user. Here, the user can query our database with many different types of preferences:

welcome to our lodging and dining service, sam!  
 Tell us a bit about your perfect vacation...

Lodging Price:

\$0 - \$410

Lodging Type:

☐ Apartment
☐ Condo
☒ Loft
☐ House
☒ Hotel
☐ Hostel
☐ Other
☐ Toggle All

Minimum # of nearby restaurants:

8

Categories I like:

mexican, greek

Restaurant Price:

\$\$

Room Type:

Entire Home / Apt

Preferred Neighborhood:

Number of Travelers:

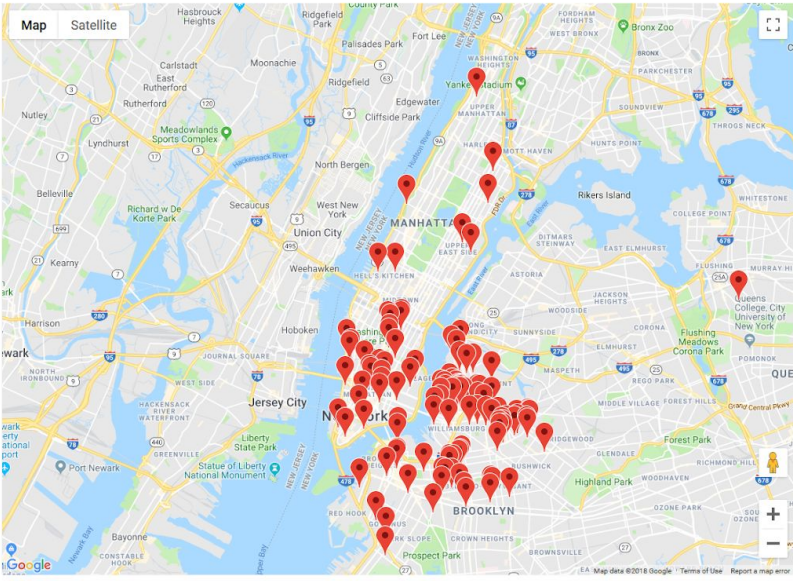
3

Restaurant Search Radius:

0.3

Miles

Search



### Our top recommendations for you:

~**GORGEOUS DOWNTOWN LOFTS**~ MULTIPLE SPACES	
Price per night: 300   Average Rating: 96   Our rating for you: 230	
Unique sunny loft in Manhattan!	
Price per night: 249   Average Rating: 92   Our rating for you: 224	
New Museum 2 BR Loft	
Price per night: 395   Average Rating: 97   Our rating for you: 223	
⚡Quiet Gem on NY's Hottest Street⚡	
Price per night: 395   Average Rating: 100   Our rating for you: 197	
CHARMING NYC Studio Loft NEAR SOHO!	
Price per night: 136   Average Rating: 97   Our rating for you: 195	

As you can see, we consider price range of vacation stays. Users can query our database here with options such as price range of both lodging and restaurants. In addition, we also consider options such as lodging type, room type, number of travelers, minimum number of nearby restaurants, food preferences, preferred neighborhood, and the restaurant search radius. On the bottom of this page, we have a recommendations tab that shows the top search results we rated for you. These recommendations are based on our rating system, which rates suitable housing depending on whether the restaurants nearby cater to your food preferences.

Food For Thought

My Account

Start Your Journey!

Sign Up

Logout

jonku

\*\*\*\*\*

Login

Sign Up for a New Account

Username:

Name

Email:

Email

Password:

Password

Password Repeated:

Password

Sign Up

After a visitor signs up, he/she will be able to see his/her name in the main search page, and will also be able to plan a trip. By clicking on a suitable Airbnb recommendation on the search page after entering preferences, he/she will be redirected to a new page that allows him/her to save a planned trip. The page shows local restaurants nearby the Airbnb, which the user can check off to save to a planned trip. An additional box allows the user to enter comments about other places the user would like to go.

Food For Thought

My Account

Start Your Journey!

Sign up

Logout

Username

Password

Login

Spacious light tribeca loft!

Tribeca Loft (Entire home/apt)

If you are interested in taking a vacation to this airbnb, make sure to save the trip info down below. Check off any restaurants you would be interested in visiting to save them as well. Enter any additional comments you may have of places you would like to visit!

Billy's Bakery

\$\$ | Bakeries | Coffee

3 Stars

+1(212)-647-9958

☐ Include this restaurant in my trip plans!

Tetsu

| Japanese | Sushi

4.5 Stars

+1(212)-207-2370

☐ Include this restaurant in my trip plans!

M1-5 Lounge

\$\$ | Lounges | DanceClubs | Newamerican

3 Stars

+1(212)-965-1701

☐ Include this restaurant in my trip plans!

Aire Ancient Baths

\$\$\$ | Spas | Massage

4 Stars

+1(646)-878-6174

☐ Include this restaurant in my trip plans!

Comment:

Save Travel Plan



Finally, we have our “My Account” page that records a user’s saved trips. This page queries our user database (MongoDB) to search for the user’s planned trips. Each user can have multiple planned trips, and each planned trip can have multiple restaurants the user is planning to visit. The users database keeps track of all planned trips the user has saved. Thus, this page queries embedded documents inside the user document. See below:

The screenshot shows a web application interface for a user's account. At the top, there is a navigation bar with links: "Food For Thought", "My Account", "Start Your Journey!", "Sign Up", and "Logout". To the right of these links are input fields for "username" (containing "jonku") and "password" (containing "\*\*\*\*\*"), followed by a "Login" button. Below the navigation bar, the user's information is displayed: "Username: jon123" and "Email: jonku@seas.upenn.edu". Underneath this, the heading "Your planned trips:" is followed by two trip cards. The first card is titled "large 1 BR in heart of East Village" and lists "Huertas 4 \$\$ +1(212)-228-4490" and "Organic Grill 3.5 \$\$ +1(212)-477-7177", with an additional comment: "Additional Comments: Make sure to also check out Shinbashi". The second card is titled "Great room in a 2 BD apartment" and lists "Shawarmania 4 \$ +1(718)-545-1890", with an additional comment: "Additional Comments:". The background of the page is a light blue gradient.

## 6. Look and Feel

We used CSS, bootstrap, and handlebars to make the look and feel of our webpages comfortable to users.

## 7. Complex SQL Queries

By far the most complicated SQL queries occur on the home page, after the user enters their requested features and presses the “search” button. This query has a WHERE clause that filters Airbnbs with the specified properties (e.g. room type, property type, etc.). It has a JOIN with the Yelp dataset on a somewhat complicated join key (which compares geographical coordinates). It aggregates on Airbnbs to generate fields like NUM\_RESTAURANTS, which we generate with COUNT(\*) on the aggregated relation. It uses a CASE WHEN statement to assign a “score” to each restaurant based on the user’s preferences -- 3 if the restaurant’s description includes a keyword that the user said he/she likes, 0 if it includes a keyword that the user said he/she dislikes, and 1 otherwise. This “restaurant score” is then fed into the SUM function, which gives us aggregate ranks for each airbnb, where rank is given as the sum of scores of nearby restaurants. There is another CASE WHEN statement which assigns a value of 1 to airbnbs in the user’s preferred neighborhood and 0 to all others. We also have a HAVING clause to filter out Airbnbs without enough nearby restaurants. Lastly we have a GROUP BY clause that ranks Airbnbs first by whether or not they’re in the user’s preferred neighborhood, and then by their rank (i.e. the sum of nearby restaurant scores).



In summary, this query has aggregation (with 2 different functions on the aggregation -- count and sum), two case-when clauses, a group by clause (with 2 attributes to sort on), a having clause, and a somewhat complex join.

#### *8. Consideration of Performance*

We used indexing to improve performance. Specifically, we found indexes on latitude and longitude to be beneficial in calculating joins, and an index on neighbourhood was beneficial since we often sorted results by neighbourhood first, then by our ranking.

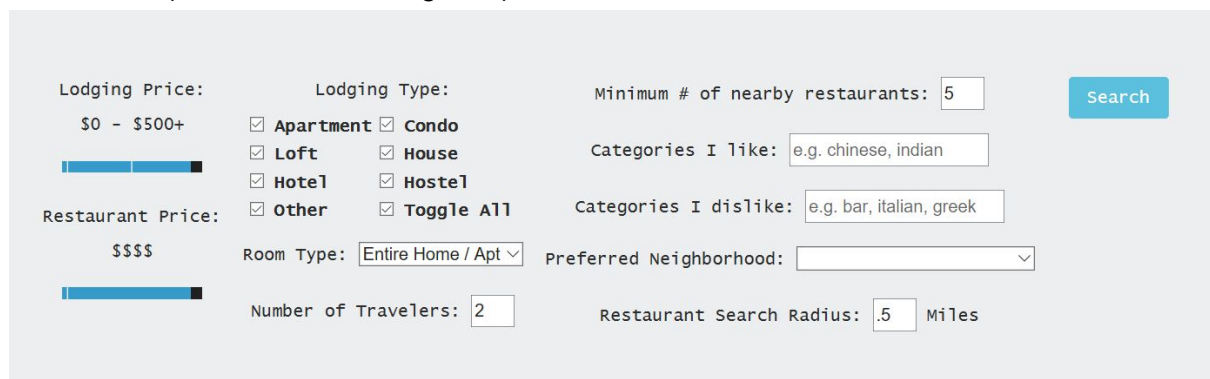
Another critical feature we implemented with performance in mind is the "NEARBY" table. This is essentially a cache which we expected to save time in calculating joins on geographical coordinates. Because many users would input their search with similar fields -- particularly, many users wouldn't bother to change the "restaurant search range" field -- it would be redundant to calculate the same join several times.

## Performance evaluation

Time cost of several example queries:

### Example 1:

Home page search, using these parameters, with and without indexing on airbnb and yelp coordinates (i.e. latitude and longitude)



The screenshot shows the Airbnb search interface with the following parameters set:

- Lodging Price:** \$0 - \$500+ (represented by a blue bar)
- Lodging Type:** Apartment, Condo, Loft, House, Hotel, Hostel, Other, and Toggle All are all checked.
- Minimum # of nearby restaurants:** 5
- Categories I like:** e.g. chinese, indian
- Categories I dislike:** e.g. bar, italian, greek
- Restaurant Price:** \$\$\$\$ (represented by a blue bar)
- Room Type:** Entire Home / Apt
- Preferred Neighborhood:** (empty dropdown)
- Number of Travelers:** 2
- Restaurant Search Radius:** .5 Miles

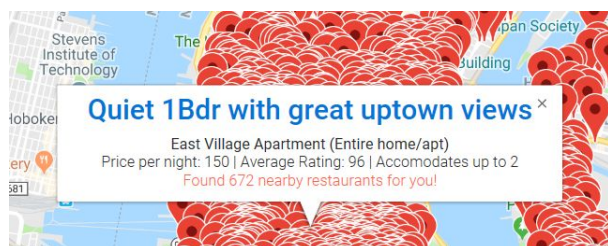
A blue "Search" button is located on the right side of the form.

```
SELECT ADR.LATITUDE, ADR.LONGITUDE, A.NAME, A.ID, A.PRICE, ADR.NEIGHBOURHOOD, A.REVIEW_SCORES_RATING, A.ACCOMMODATES,
       A.PROPERTY_TYPE, A.ROOM_TYPE, SUM (CASE WHEN (Y.CATEGORIES LIKE '%undefined%') THEN 3 WHEN (Y.CATEGORIES LIKE
'%undefined%') THEN 0 ELSE 1 END) AS RANKING, CASE WHEN ADR.NEIGHBOURHOOD = 'undefined' THEN 1 ELSE 0 END AS GOODLOC, COUNT(*) AS
NUM_RESTAURANTS FROM AIRBNB A JOIN AIRBNB_ADDRESS ADR ON A.ID = ADR.ID JOIN YELP_DATA Y ON ADR.LATITUDE - Y.LATITUDE >
-0.007246376811594203 AND ADR.LATITUDE - Y.LATITUDE < 0.007246376811594203 AND ADR.LONGITUDE - Y.LONGITUDE > -0.009615384615384616 AND
ADR.LONGITUDE - Y.LONGITUDE < 0.009615384615384616 WHERE ACCOMMODATES >= 2 AND Y.PRICE <= 4 AND A.PRICE <= 500 AND ROOM_TYPE = 'Entire
home/apt' AND ( PROPERTY_TYPE = 'Apartment' OR PROPERTY_TYPE = 'Condominium' OR PROPERTY_TYPE = 'Loft' OR PROPERTY_TYPE = 'House' OR
PROPERTY_TYPE = 'Hotel' OR PROPERTY_TYPE = 'Hostel' OR (PROPERTY_TYPE != 'Apartment' AND PROPERTY_TYPE != 'Condominium' AND
PROPERTY_TYPE != 'Loft' AND PROPERTY_TYPE != 'House' AND PROPERTY_TYPE != 'Hostel' AND PROPERTY_TYPE != 'Hotel' ) ) GROUP BY ADR.LATITUDE,
ADR.LONGITUDE, A.NAME, A.ID, A.PRICE, ADR.NEIGHBOURHOOD, A.REVIEW_SCORES_RATING, A.ACCOMMODATES, A.PROPERTY_TYPE, A.ROOM_TYPE
HAVING COUNT(*) >= 5 ORDER BY GOODLOC DESC, RANKING DESC
```

Without indexing: 25789.448 ms, or over 25 seconds!!

With indexing: 22479.768 ms -- over 22 seconds.

We see that indexing improves performance here, but not by much unfortunately! We suspect that this is because the restaurant search radius is large enough that each airbnb is joined with a very large number of restaurants. A quick scan over the results returned shows that some airbnbs returned well over 500 nearby restaurants (as shown below)!



### Example 2:

Home page search, using these parameters, with and without indexing on airbnb and yelp coordinates (i.e. latitude and longitude)

Lodging Price: \$0 - \$280  
Restaurant Price: \$\$  
Lodging Type: ☐ Apartment ☐ Condo ☐ Loft ☒ House ☐ Hotel ☐ Hostel ☐ Other ☐ Toggle All  
Room Type: Entire Home / Apt  
Number of Travelers: 2  
Minimum # of nearby restaurants: 5  
Categories I like: mexican, italian  
Categories I dislike: japanese  
Preferred Neighborhood: Upper East Side  
Restaurant Search Radius: .2 Miles  
Search

```
SELECT ADR.LATITUDE, ADR.LONGITUDE, A.NAME, A.ID, A.PRICE, ADR.NEIGHBOURHOOD, A.REVIEW_SCORES_RATING, A.ACCOMMODATES,
       A.PROPERTY_TYPE, A.ROOM_TYPE, SUM (CASE WHEN (Y.CATEGORIES LIKE '%mexican%' OR Y.CATEGORIES LIKE '%italian%') THEN 3 WHEN
(Y.CATEGORIES LIKE '%japanese%') THEN 0 ELSE 1 END) AS RANKING, CASE WHEN ADR.NEIGHBOURHOOD = 'Upper East Side' THEN 1 ELSE 0 END AS
GOODLOC, COUNT(*) AS NUM_RESTAURANTS FROM AIRBNB A JOIN AIRBNB_ADDRESS ADR ON A.ID = ADR.ID JOIN YELP_DATA Y ON ADR.LATITUDE -
Y.LATITUDE > -0.002898550724637681 AND ADR.LATITUDE - Y.LATITUDE < 0.002898550724637681 AND ADR.LONGITUDE - Y.LONGITUDE >
-0.0038461538461538464 AND ADR.LONGITUDE - Y.LONGITUDE < 0.0038461538461538464 WHERE ACCOMMODATES >= 2 AND Y.PRICE <= 2 AND A.PRICE <=
280 AND ROOM_TYPE = 'Entire home/apt' AND ( PROPERTY_TYPE = 'House' ) GROUP BY ADR.LATITUDE, ADR.LONGITUDE, A.NAME, A.ID, A.PRICE,
ADR.NEIGHBOURHOOD, A.REVIEW_SCORES_RATING, A.ACCOMMODATES, A.PROPERTY_TYPE, A.ROOM_TYPE HAVING COUNT(*) >= 5 ORDER BY
GOODLOC DESC, RANKING DESC
```

Without indexing: 3273.614 ms -- just over 3 seconds.

With indexing: 771.013ms

That's an improvement by roughly a factor of 5! By narrowing the search radius to .2 miles, we've made the possible "nearby" region smaller by a factor of 6 -- i.e.  $.05^2 / .02^2$ .

### Example 3:

Lastly, let's consider an example where we use the NEARBY table instead of calculating the join on-the-fly. This is not for direct comparison with the examples above, but rather shows how indexing affects performance when we use the cache.

```
SELECT ADR.LATITUDE, ADR.LONGITUDE, A.NAME, A.ID, A.PRICE,  
A.NEIGHBOURHOOD, A.REVIEW_SCORES_RATING, A.ACCOMMODATES,  
A.PROPERTY_TYPE, A.ROOM_TYPE, SUM (CASE WHEN (Y.CATEGORIES LIKE  
'%chinese%') THEN 3 WHEN (Y.CATEGORIES LIKE '%bar%') THEN 0 ELSE 1 END) AS  
RANKING, CASE WHEN A.NEIGHBOURHOOD = 'Park Slope' THEN 1 ELSE 0 END AS  
GOODLOC, COUNT(*) AS NUM_RESTAURANTS FROM AIRBNB A JOIN  
AIRBNB_ADDRESS ADR ON A.ID = ADR.ID JOIN NEARBY N ON A.ID = N.AIRBNBID  
JOIN YELP_DATA Y ON N.YELPID = Y.ID WHERE ACCOMMODATES >= 5 AND  
ROOM_TYPE = 'Entire home/apt' AND ( PROPERTY_TYPE = 'Loft' ) GROUP BY  
ADR.LATITUDE, ADR.LONGITUDE, A.NAME, A.ID, A.PRICE, A.NEIGHBOURHOOD,  
A.REVIEW_SCORES_RATING, A.ACCOMMODATES, A.PROPERTY_TYPE,  
A.ROOM_TYPE HAVING COUNT(*) >= 5 ORDER BY GOODLOC DESC, RANKING DESC
```

Before indexing: 435.195 ms

After indexing: 400.055 ms

As we see, indexing offers little improvement when we have the NEARBY table.

## Technical Challenges & How They Were Overcome

The biggest technical challenge on the front end was getting the Oracle database connection to work. Initially, we ran into connection issues after it seemed like the database was set up correctly. In order to connect to our Oracle database, we had to download an instant client, and the connection problem seemed to be arising from it. We spent multiple hours trying to figure out what was happening, and finally we realized that it was due to compatibility issues with certain versions of npm on a Mac. We downgraded our npm versions, and it started working magically. This taught us that setting up the Oracle database was much harder than setting up a database such as MongoDB.

Another technical front-end challenge we ran into was the integration of handlebars, angular, and bootstrap. When trying to integrate them together, certain aspects would break and we would need to fix them. This often got in the way of programming our actual website functionalities. Problems such as handlebars not generating layouts correctly, routers not redirecting to the correct page, and handlebars layouts not generating correctly would often appear. We spent a good portion of our time working on the page layout bugs. Ultimately, we overcame this by splitting up the work by searching online for fixes. This taught us how valuable online resources are when learning how to program using new tools.

The final worth-mentioning front-end technical challenge we ran into was conflicts in running queries on the backend database. When we split up into front-end and back-end teams, we tried our best to communicate our work so that integration would be easier. However, since we were working as separate teams, sometimes problems arose because we had differences in our thought processes. For example, when back end worked on storing Yelp and Airbnb price ranges, it was initially stored as a String. However, because prices were stored as Strings instead of Integers, when we went to query the database for price ranges less than or equal to a certain price, it would return incorrect results. We spent a good time figuring out if we were doing things wrong on the front-end that led to these results. However, we finally realized it was a conflict with the back end storage of the price when discussing this with back-end. We were able to overcome and avoid potential obstacles such as these by maintaining constant communication with back-end.

One technical challenge on the backend is to connect the database to AWS. There are lots of configuration when we set up the Oracle database on AWS, after creating the database on AWS, we need to connect it with our local Oracle SQL developer. At first, we cannot connect it with AWS until finish reading the whole documentation online and change the security settings.

Another technical challenge on the backend is to store the database to local Oracle. At first, we tried to use python to create engine and put the dataframe in the Oracle, however, there is some data type error, after converting the data type, we found out it would take a long time to store the data into AWS since the database is too large, then we searched online and tried several methods, such as store as sqlite format first and convert to sql ddl. Finally, we found out there would always be some data type error, so we decided to use python store all the clean data as csv file, and then simply import the csv file into Oracle, at the same time, we can deal with the datatype and columns when importing the file.

### **Extra Credit Features**

1. Use of MEAN stack
2. Use of NoSQL
3. Use of cloud hosting (AWS & mLab)
4. Integration with Google Maps