FoodComm: A Social Media Revolution powered by Neo4J

By
Jesse Dong,
Samyak Jagdish Kumbhalwar, and
Lilou Sicard-Noel

For CS 157C - NoSQL Databases
With Dr. Wu
Spring 2023

Table of Contents

Chapter 1. Project Description

History:

Loneliness in young adults has been on the rise and increasing steadily since 1976. According to the article, “Is loneliness in emerging adults increasing over time?” today’s young adults are lonelier than young adults from previous decades. Loneliness is an issue that has a variety of different adverse effects. Not only are individuals affected, but there is also a high economic cost to society.

One major problem with social media apps is that people can connect and make friends with anyone worldwide. It is rare that these friendships ever evolve from the Internet into real life. Our application aims to address the issues of traditional social media and turn online connections into lifelong friendships.

Goal:

FoodComm aims to create a social network where users can interact and share food preferences. Our platform intends to bring people together while also giving great restaurant recommendations.

FoodComm is a database application similar to Facebook, Twitter, and Yelp applications. Once a user has made an account, they can add connections and build their network. They can also review restaurants and add their favorites to their profiles. Connected users will then be able to see each other's preferences and will be suggested new restaurants based on similar likings. Other information, such as allergies and dietary preferences, will also be available.

Motivation:

Eating alone can be a daunting task. The motivation of FoodComm is to allow people to make friends through a common interest in food. Furthermore, FoodComm also helps individuals who struggle to choose where to eat. Though the primary motive for FoodComm is to help reduce loneliness, through the recommendation feature, users will also spend less time thinking about what to eat.

Stakeholders:

FoodComm’s stakeholders include:

- Users
- Restaurants
- Restaurants owners/workers

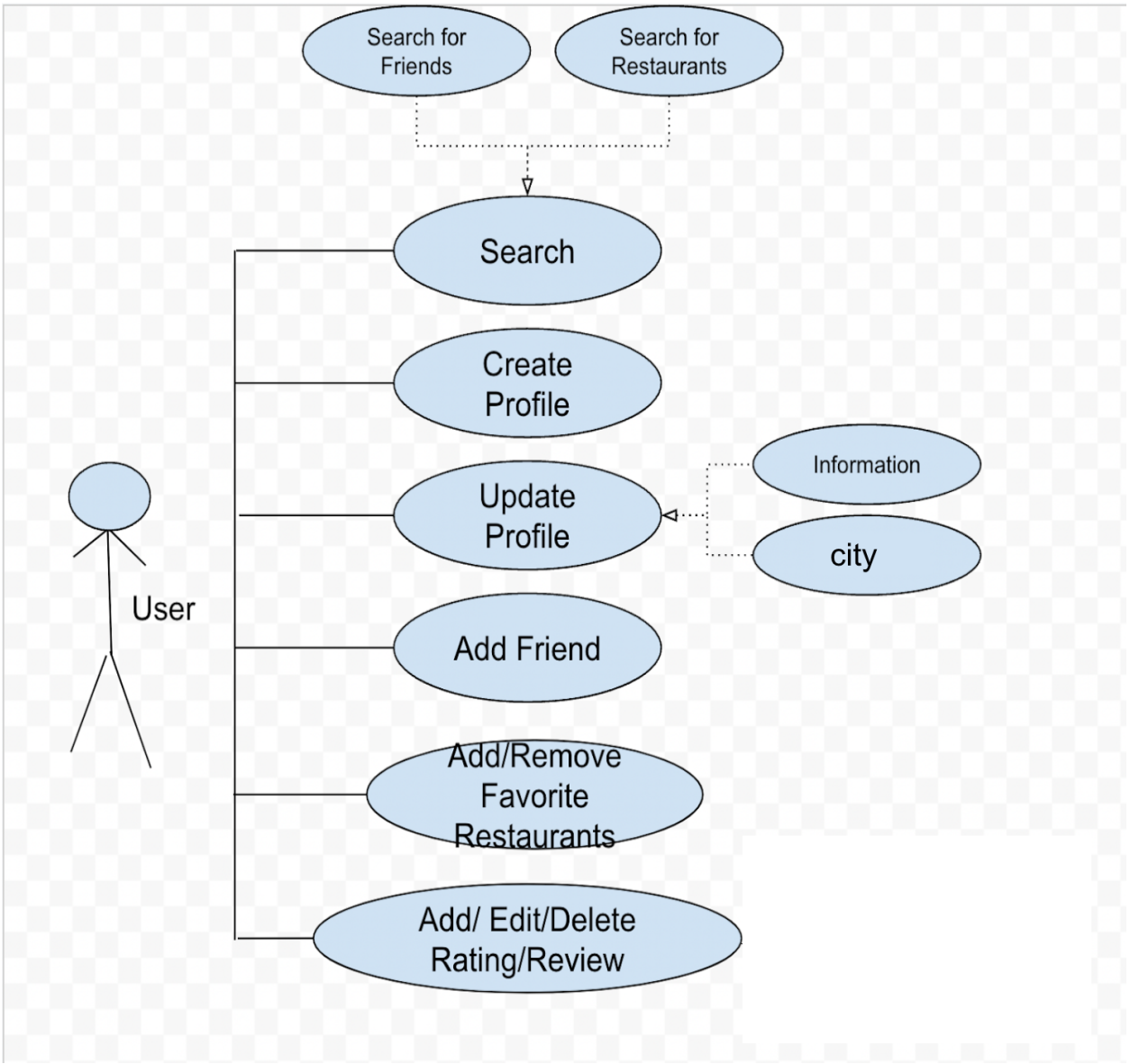
Application Domain:

The initial release of FoodComm will be available to San Jose State University students, consisting of around 35,000 students. Once a solid user base is established, we will branch out to

the city of San Jose. As we grow, we plan to expand our domain statewide, countrywide, and worldwide.

Chapter 2. Project Requirements

Use Case Diagram:



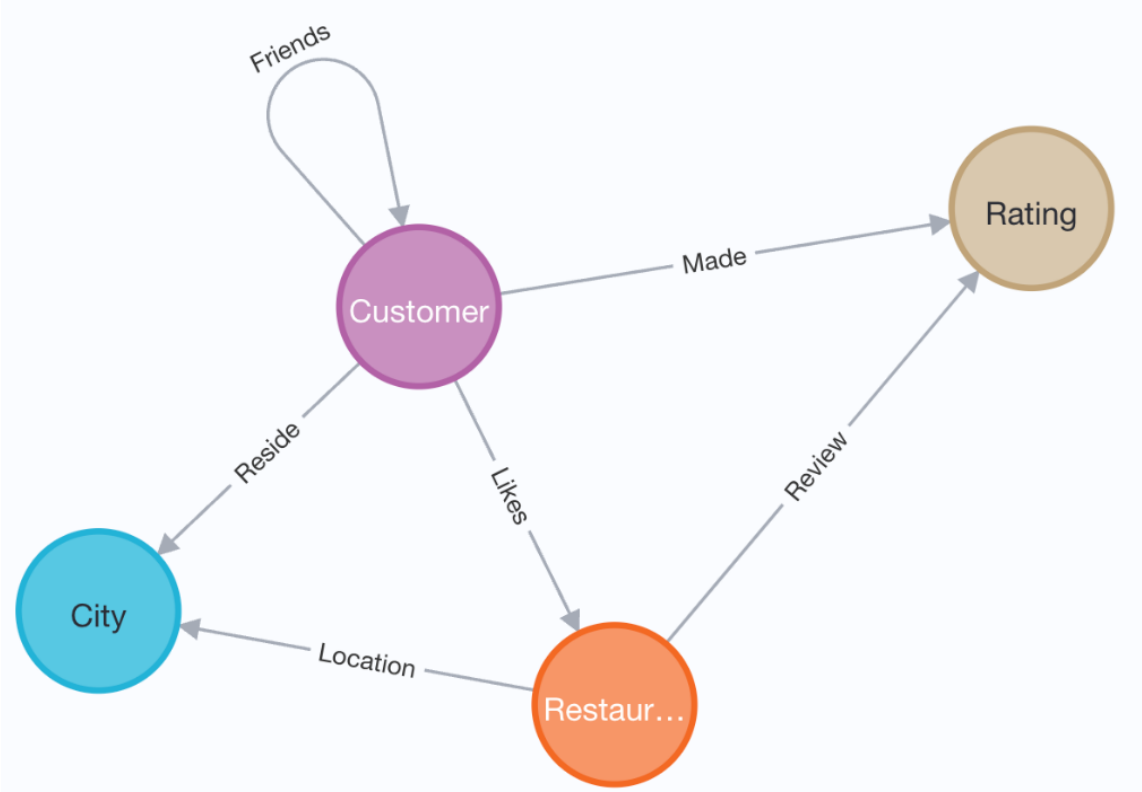
Chapter 3. Project Design

NoSQL Database Used

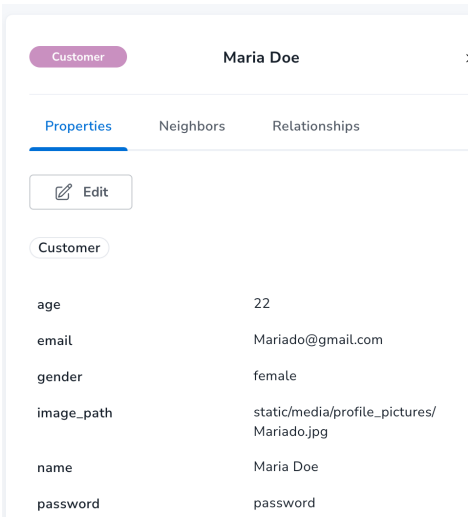
For this project, Team 6 was assigned Neo4j. Neo4j is a graph database; therefore, making a project where the relationship between objects is a priority would make sense. This is why Team 6 decided to make FoodComm, a social media web application. Making a social media with a graph database combines the advantages of a relational database with the advantages of the NoSQL database.

Node Types

FoodComm's database is a graph that consists of four types of nodes and six types of relationships.

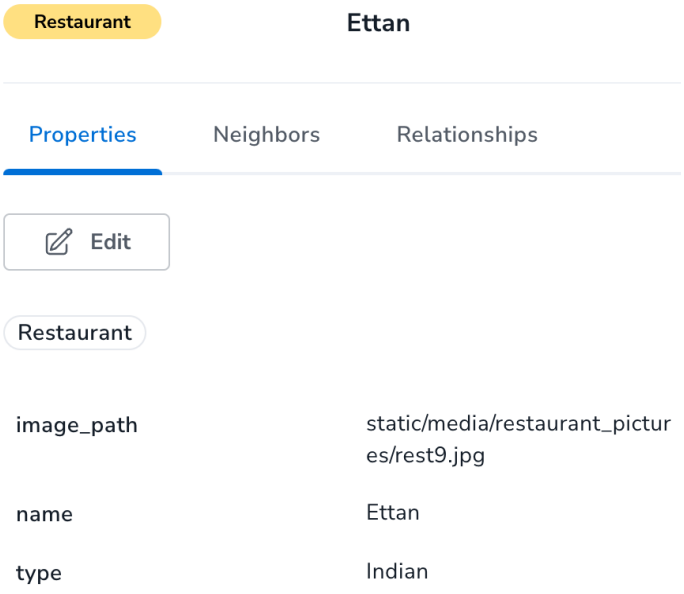


The Customer node is the central object of the database. The node itself has several attributes: the name of the customer, the email, password, age, gender, and the user's profile picture path.

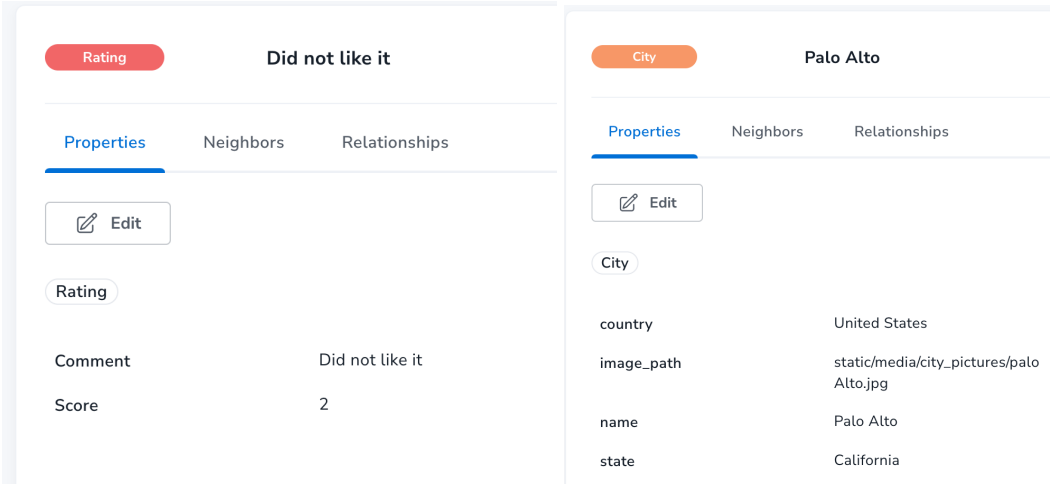


The Made relationship indicates that the Customer is the author of a specific Rating node. The Friends relationship indicates who are the other customer nodes this node is friends with. The Friends relationship has created both ways when a user adds a friend (The user's node points to that new friend, and the friend's node points to the user's node). The Customer also Resides in a city, allowing the web application to offer recommendations based on the location of the user. Finally, the Customer can Like a Restaurant, allowing the Customer to find this restaurant and improve their friends' recommendations.

The Restaurant node is the other important node of this database. The restaurant node points to the city where it is located and to all the review nodes that were created for the restaurant.



Finally, the city and review nodes contain information about their appropriate content. A city has a name, state, country, and image path. The review has the rating and the comment.



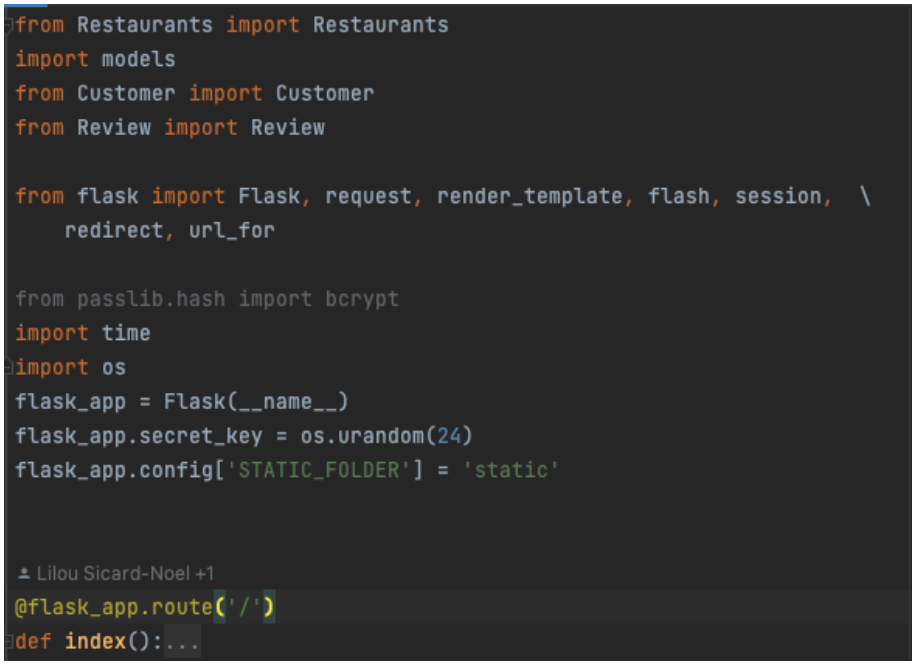
(Please see Appendix 1 for a screenshot of all the entities in the database)

Chapter 4. Project Implementation

Our project was implemented using two main Python libraries, Flask and py2neo. Flask worked as our web framework, and py2neo was the library used to connect and interact with our database. The main files for our project are main.py, models.py, Customer.py, Restaurants.py, and Review.py. Since the code for our project can be seen on GitHub, we won't be showing all the screenshots here as it's simply too much. Instead, we will go through our website's features below and provide a basic overview of how our code works.

Main.py

This module is the heart of our project. This is where Flask has been set up and how the layout of our website is determined.



```
from Restaurants import Restaurants
import models
from Customer import Customer
from Review import Review

from flask import Flask, request, render_template, flash, session,  \
    redirect, url_for

from passlib.hash import bcrypt
import time
import os
flask_app = Flask(__name__)
flask_app.secret_key = os.urandom(24)
flask_app.config['STATIC_FOLDER'] = 'static'


 Lilou Sicard-Noel +1
@flask_app.route('/')
def index():...
```

Above is a snippet of the Main.py module. As you can see, all the other modules have been imported. Furthermore, you can see how our application is built. The '/' is the default page when the user enters the webpage. So we have defined a method called index that tells our application

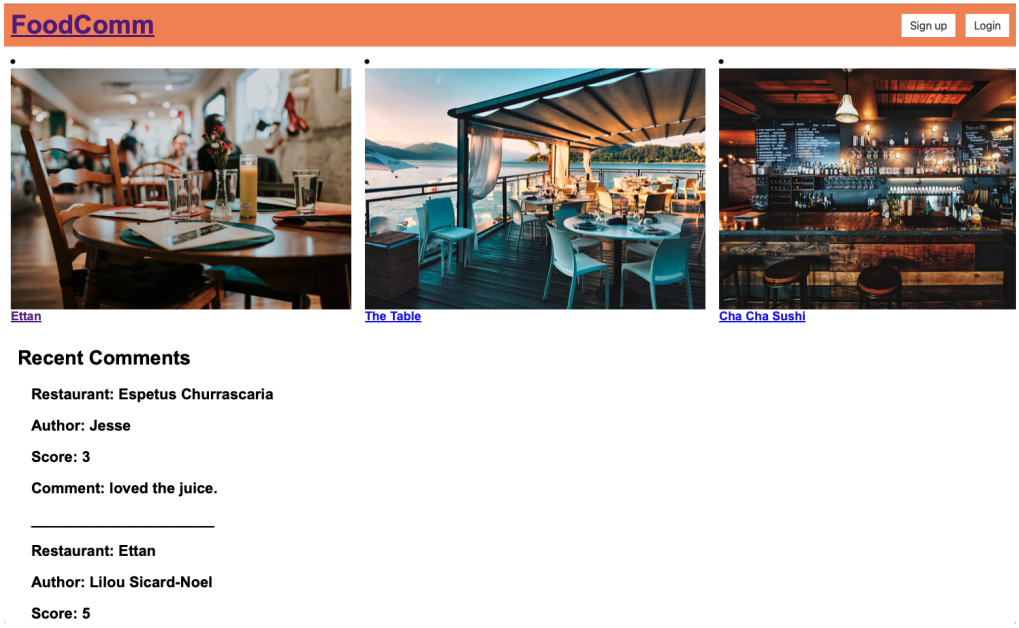
what to do on that page. Below are more screenshots of other pages.

The functionality and specifics of each page will be covered later in this section. Finally, we have a main section that simply starts our flask app.

Below we will cover our website functionalities and how they're implemented. Since it is not practical to screenshot all the code, we'll mainly focus on the database itself. All the code is available to see on Github: https://github.com/lilousicard/CS157C-Team6. For the Index page, we'll show the code so you can see the structure of our application. All the pages follow the same pattern as the Index page.

Index.html

This is our website's homepage, as shown below.



As you can see, before a user is logged in, they can see a list of restaurants as well as any comments that other users have made. Below is how this page is rendered.

```
@flask_app.route('/')
def index():
    rests = Restaurants("")
    restaurants = rests.get_all()
    rev = Review("", "", "", "")
    reviews = rev.get_all_review()
    return render_template('home.html', list=restaurants, reviewList=reviews)
```

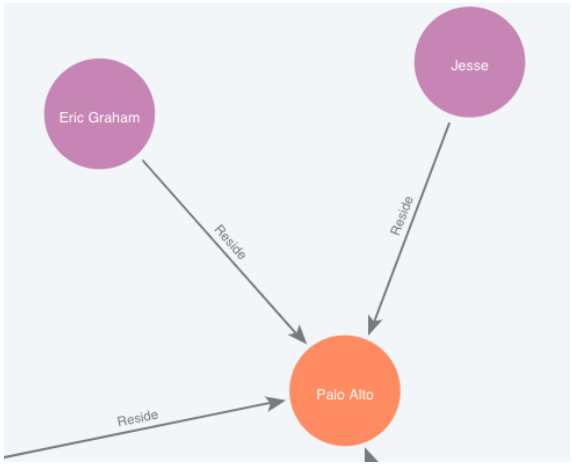
Methods such as rests.get_all() and rev.get_all_review() are defined in each respective modules Restaurants.py and Review.py. Then these results are passed into our home.html file.

```
{% extends "base.html" %}
{% block head %}
    {{ super() }}
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='base.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='home.css') }}">
{% endblock %}
{% block content %}
    <main>
        <section class="restaurant-list">
        {% if list %}
          {% for rests in list [:10] %}
            <li>
              <div class="rest-item">
                <div class="rest-details">
                  <img src = {{rests.get('image_path')}}/>
                  <h3> <a href = "restaurant/{{ rests.get('name') }}">{{ rests.get('name') }}</a></h3>
                </div>
              </div>
            </li>
          {% endfor %}
         {% endif %}
        </section>
```

This snippet shows our home.html file and how the restaurant lists are rendered. The full code is available on GitHub. For the rest of our project, each page follows this exact pattern. We define and call needed methods. Those results are then passed into the HTML files to be rendered. If you're interested in the exact details of how each other page is rendered, please refer back to the GitHub link.

Register

A user is asked to fill out a few forms on our register page before their account is created. These forms are self-explanatory. We use the customer's email as a primary key. Thus there can only be one customer node per email to avoid duplication issues. As for the city, if a user enters a city that already exists inside our database, we simply create a relationship of "Reside" to that city node from our new user node. If the city node doesn't exist, we create a new city node first, then create the same "Reside" relationship. A user enters the register page by clicking the signup button. Below is what our graph looks like for customers and cities.



Below are the attributes of Customer and City nodes, as well

Customer

age	21
email	jesse@gmail.com
gender	male
image_path	static/media/profile_pictures/122284172-smiling-banana-cartoon-kawaii-character-banana-fruit-vector-isolated-illustration-green-background.jpg
name	Jesse
password	123456

City

country	United States
image_path	static/media/city_pictures/palo Alto.jpg
name	Palo Alto
state	California

Login

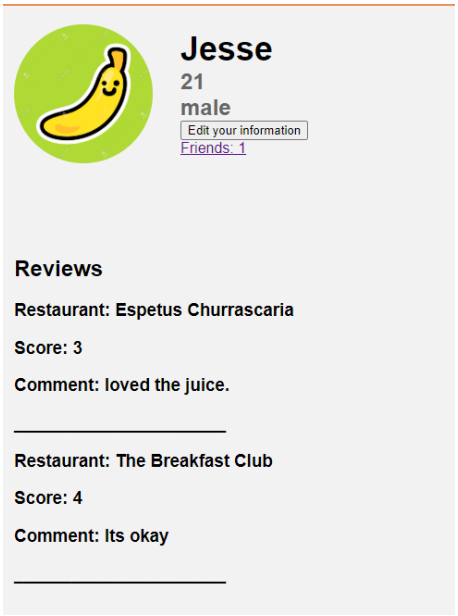
The login function works by searching for the node given a customer's email address and checking if the inputted password is the same as the password stored in the database. Using py2neo's library, we can easily search for this, as shown below. Also, as you saw from the customer node earlier, current security is not the greatest, as passwords are not encrypted. We have left it this way for testing purposes. For deployment, passwords would 100% be encrypted using Python's Bcrypt library. This would work by passing the password into Bcrypt upon registration. Then during the login process, as you can see in the commented-out code, we would once again pass in the inputted password into Bcrypt to test if they're the same.

```python
class Customer:
    Jesse2131
    def __init__(self, email):
        self.email = email

    Jesse2131
    def find(self):
        cust = matcher.match("Customer", email=self.email).first()
        return cust

    Jesse2131
    def verify_password(self, password):
        cust = self.find()
        if cust:
            return password == cust["password"]
            # return bcrypt.verify(password, cust["password"])
        else:
            return False
```

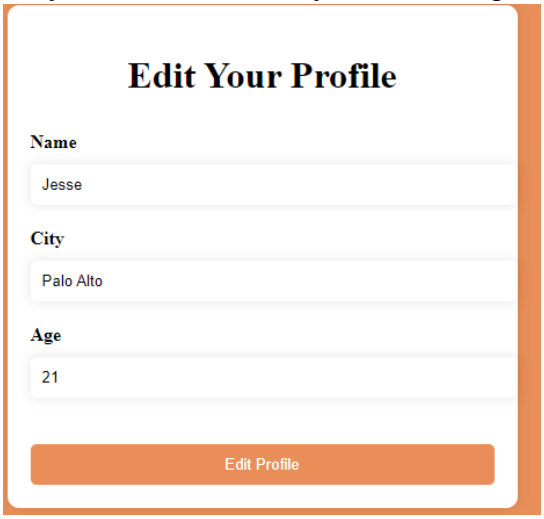
```python
@flask_app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')
        if not Customer(email).verify_password(password):
            flash('Invalid login.')
            return render_template('login.html', error=True)
        else:
            session['user'] = email
            flash('Logged in.')
            return redirect(url_for('index'))

    return render_template('login.html')
```

User Profile

Once the user is logged in, they can access their profile page. This page displays basic information such as their name, age, and friend count. There is also a section that shows any reviews that they have made.

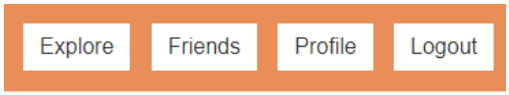


They also have the ability to edit their profile and change their name, age, or city



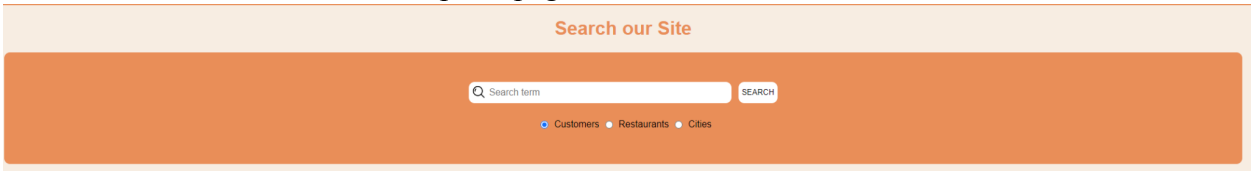
Explore

After logging in, users are returned to the index page, as shown previously. The navigation bar will now give them more options, specifically to look at the explore page and a friends button.



Search

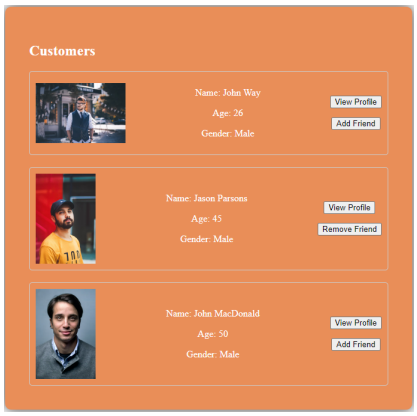
One of the main features of the Explore page is the search bar.



Here users can search for the different types of nodes in our database. The search is done by using a cypher query and matching the search term to the node's attributes. It's important to note that sensitive attributes such as email and password are ignored in this search.

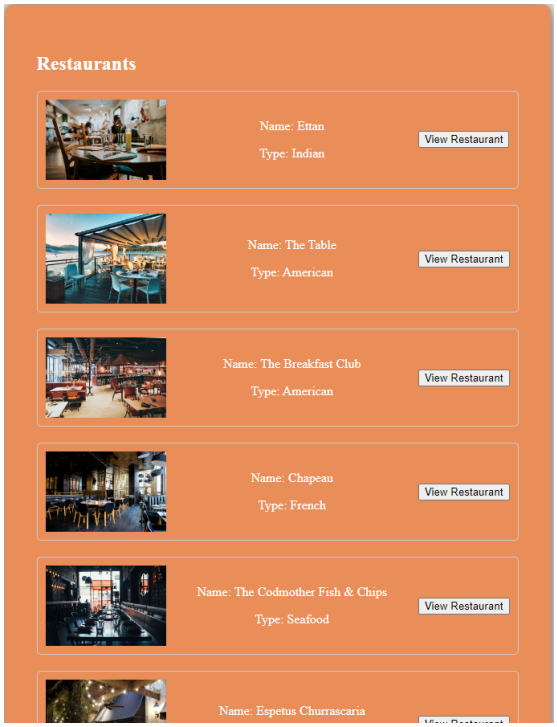
```
Jesse Dong +1
def search_node(node, search_term, exclude_term):
    target_node = node.capitalize()
    # search for nodes that have attributes that match the search term,
    # but exclude password, email, and image path
    results = matcher.match(target_node).where(
        f"any(attr in keys(_) where attr <> 'password' and attr <> 'email' "
        f"and attr <> 'image_path' and _["f"attr]  =~ '("f"?i)."
        f"*{search_term}.*')"
    )

    # if searching for users, exclude the currently logged-in user
    if target_node == "Customer" and exclude_term is not None:
        results = results.where(f"_.email <> '{exclude_term}'")

    return [dict(node) for node in results]
```

Then, users are taken to the search results page, where nodes that have matched the result are displayed. For example, if the search term is simply the letter “j,” customers with j in their name will be shown.

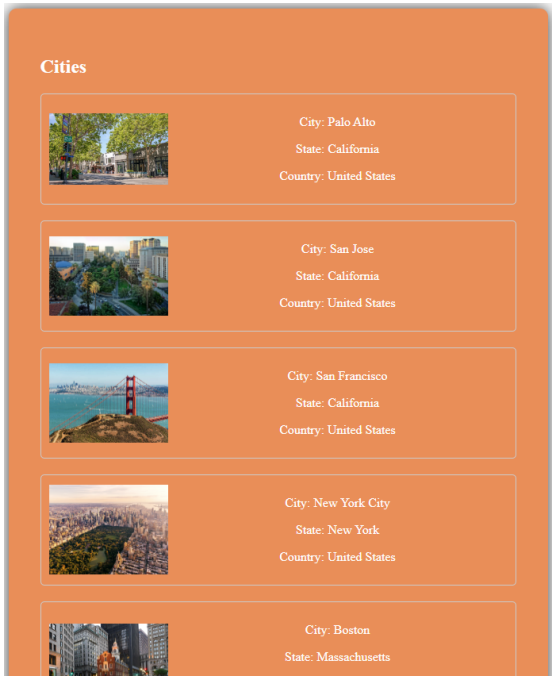


As you can see, from here, a user can view their profiles and add/remove them from their friendslists. Searching for restaurants/cities return a similar view.

Restaurant:



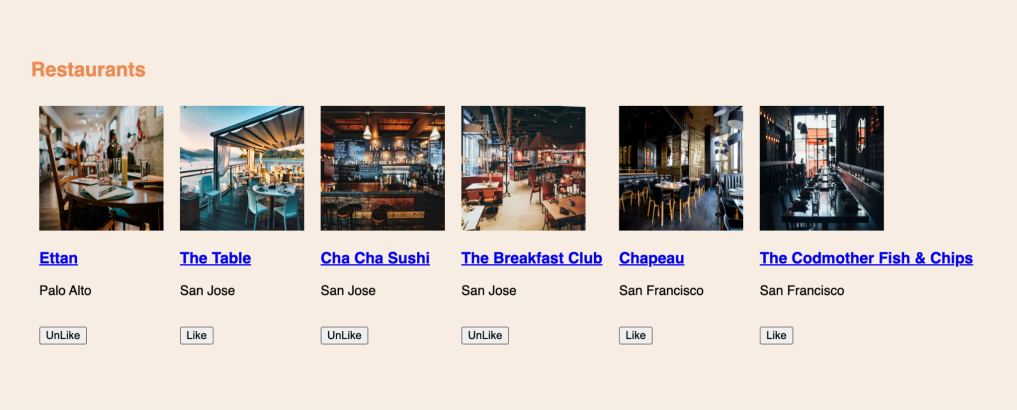
City:



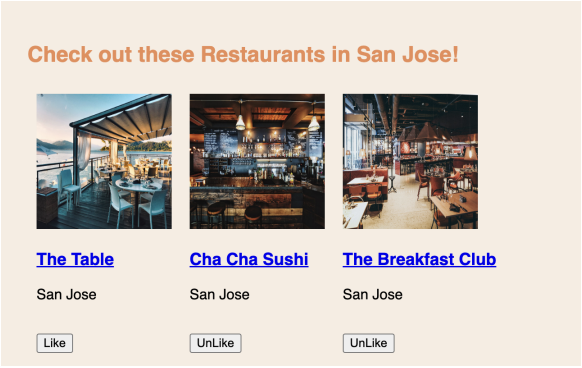
Suggested Restaurants

The Explore page is designed for users to explore and find new restaurants. The idea is to utilize the graph database power and recommend restaurants based on their activity to users. Users are also provided with a like button below each restaurant.

1. Trending restaurants:
   Based on ratings and reviews, restaurants are recommended to users.

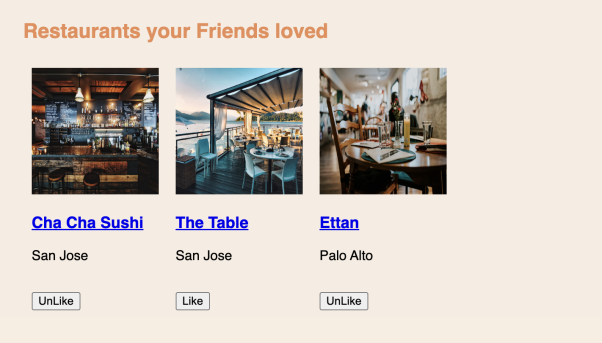


2. Restaurants based on City:
   Users, when registering, provided personal details, which included city information. Based on it, users are suggested restaurants. This is implemented by tracing out Restaurant nodes to city nodes in our graph db.



3. Restaurants based on Friends activity:
   Users also get suggestions based on their friends' activity in this application. If a user is friends with another person and if they like or reviews any restaurant, the user's explore page will incorporate this restaurant. This is implemented by traversing the graph in the following manner:

   User -> Friends -> Likes -> Restaurant.



Customer features

Customers of our site have a few main features, including adding friends, liking restaurants, and posting reviews.

Adding/Removing Friends

As seen previously, users are able to search for other users to add them as friends. In our database, a friendship is simply just a relationship of "Friends" between two customer nodes. So, whenever the add/remove friend button is clicked, the respective method is called, and we use py2neo to simply create the relationship between the nodes. One thing to note is that Neo4j relationships are not bidirectional by default. So for our implementation, we wanted both users to be friends with each other if one of them added the other as a friend. Thus it was necessary to create to "Friends" relationships between both users. The code below is how this is done and is located in the Customer.py class.

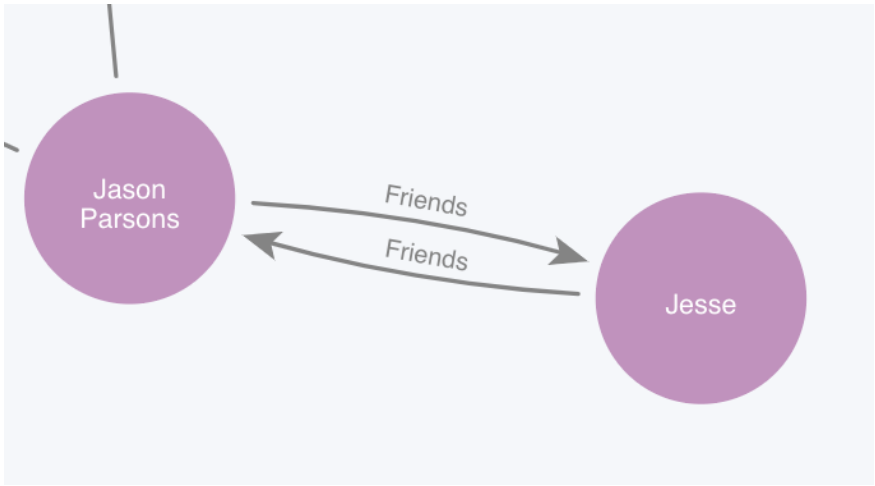
```
Jesse2131 +1
def add_friend(self, friend_email):
    cur_user = self.find()
    # search for the user that has been sent a friend request
    friend_node = models.get_customer(friend_email)
    # create connection of friend_node was found(Should be no error)
    if friend_node:
        graph.create(Relationship(cur_user, "Friends", friend_node))
    else:
        print(f"User {friend_email} doesn't exist")

Jesse Dong
def remove_friend(self, friend_email):
    cur_user = self.find()
    # search for the user that has been sent a friend request
    other_user = models.get_customer(friend_email)
    # remove connection between the two nodes
    # Don't use graph.delete, will also remove the nodes
    rel = rel_matcher.match(nodes=[cur_user, other_user],
                            r_type="Friends").first()
    graph.separate(rel)
```

As you can see, thanks to py2neo, this is very simple to do, and there is not much code at all. In this case, we create/remove the relationship between the currently logged user and the passed-in

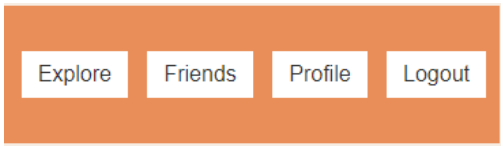
friend user. Thus, when we call these methods in the main.py module, it's necessary to call them twice for both users, as shown below.

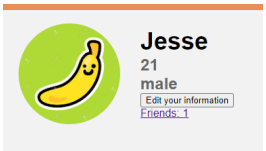
```
Jesse Dong
@flask_app.route('/add_friend', methods=["GET", "POST"])
def add_friend():
    # Relationships are not bidirectional, so need to call add friends for
    # both Customer instances
    cur_user = session.get('user')
    other_user = request.form.get('email')
    if cur_user:
        Customer(cur_user).add_friend(other_user)
        Customer(other_user).add_friend(cur_user)
        return redirect(request.referrer)
    return redirect(url_for('login'))


Jesse Dong
@flask_app.route('/remove_friend', methods=["GET", "POST"])
def remove_friend():
    # Currently, if a user unfriends someone, they will also be unfriended
    # by the other user
    cur_user = session.get('user')
    other_user = request.form.get('email')
    if cur_user:
        Customer(cur_user).remove_friend(other_user)
        Customer(other_user).remove_friend(cur_user)
        return redirect(request.referrer)
    return redirect(url_for('login'))
```

In our database, this is what two friends look like.



The user can also view their friends by clicking on the Friends button in the navbar or from their profile.





This then takes them to the Friends page, where users can see all their current friends.

Friends' Accounts

Jason Parsons

San Francisco

View Profile

Chapter 5. How to run the project

Using Docker

Our project can be run using docker as we have created a docker image. To do this, please download Docker desktop and then follow the following steps.

1. docker pull jesse24/foodcomm:latest
2. docker run -d -p **port**:5001 jesse24/foodcomm

   Replace **port** with any number preferably above 10,000 to avoid conflicting ports with your system. Here is an example

   docker run -d -p 15000:5001 jesse24/foodcomm

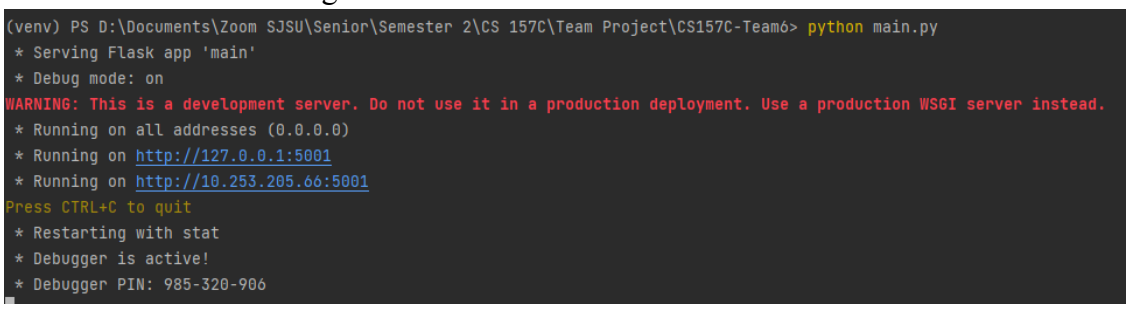
3. go to localhost:**port**

   Simply open your localhost at the port you have specified earlier. So if you used the example, go to localhost:15000

From the Github: https://github.com/lilousicard/CS157C-Team6

If, for some reason, the docker image isn't working, or you would like to run our code from the GitHub repository instead, follow these instructions. Ensure you have python3 installed and the following Python libraries: Flask, Jinja2, Py2neo, and passlib.

1. Git clone https://github.com/lilousicard/CS157C-Team6.git
2. Inside the root directly, simply run **python main.py or python3 main.py,** depending on your Python installation.
3. You will see the following:



   From here, you can go to either link, localhost, or your IP.

Chapter 6. Lesson Learned

Lilou Sicard-Noel

In this course, I learned about NoSQL databases and their application in the real world. Before this class, I had no idea that Neo4j even existed. The concept of NoSQL databases was utterly foreign to me. The essential idea I learned in this project is how to apply my knowledge of databases. For the project, this experience was a great way to improve my skills in Python and programming. It also helped me to improve my frontend skills and to discover new ways to improve my HTML documents. On a group project level, I learned how to divide the work between team members in a way that everyone can work on their greatest skill while also learning something new in the meantime.

Jesse Dong

During this project, I learned a lot about Neo4j as well as using Python to interact with the database. Specifically, I learned a lot about Py2neo, which is the python library that we used to connect to our Neo4j database. This library was crucial to the development of our project as it allowed us to implement the features of our website with relative ease. Features such as creating accounts and creating relationships could quickly be done with just a few lines of code, thanks to Py2neo. Overall, this project taught me the fundamentals of Neo4j and implemented those fundamentals in Python code to create a working web application.

Samyak Kumbhalwar

The project helped me to understand graph database and their applications in detail. It was interesting to learn more about Neo4j and its tools to interact with the graph instance. I learned about the data model and querying graph data in different ways with the help of Cypher query language. Along with working on the backend flask part, I also got to learn more about front-end development using HTML and CSS languages. Since graph db pertains to particular use cases, the project helped us identify such requirements and will help us evaluate our database choice for future projects.

Reference:

Buecker, S., Mund, M., Chwastek, S., Sostmann, M., & Luhmann, M. (2021). Is loneliness in emerging adults increasing over time? A preregistered cross-temporal meta-analysis and systematic review. Psychological Bulletin, 147(8), 787–805. https://doi.org/10.1037/bul0000332

Appendix

A1. Screenshot of all objects in the FoodComm Database

Customer Nodes


```
{
  identity: 45,
  labels: ["Customer"],
  properties: {
    password: "123456",
    gender: "male",
    image_path: "static/media/profile_pictures/122284172-smili
    name: "Jesse",
    age: "21",
    email: "jesse@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:45"
}
```

```
{
  identity: 0,
  labels: ["Customer"],
  properties: {
    password: "pa$$word",
    gender: "Male",
    image_path: "static/media/profile_pictures/eric_graham.jpg",
    name: "Eric Graham",
    age: 37,
    email: "eric.graham@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:0"
}
```

```
{
  identity: 3,
  labels: ["Customer"],
  properties: {
    password: "abcde",
    gender: "Male",
    image_path: "static/media/profile_pictures/nathan_davidson.jpg",
    name: "Nathan Davidson",
    age: 20,
    email: "nathan.davidson@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:3"
}
```

```
{
  identity: 4,
  labels: ["Customer"],
  properties: {
    password: "password",
    gender: "female",
    image_path: "static/media/profile_pictures/Mariado.jpg",
    name: "Maria Doe",
    age: 22,
    email: "Mariado@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4"
}
```

```
{
  identity: 5,
  labels: ["Customer"],
  properties: {
    password: "pa$$word",
    gender: "Male",
    image_path: "static/media/profile_pictures/JohnWay.jpg",
    name: "John Way",
    age: 26,
    email: "JohnWay@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:5"
}
```

```
{
  identity: 8,
  labels: ["Customer"],
  properties: {
    password: "a232bcde",
    gender: "Male",
    image_path: "static/media/profile_pictures/john_macdonald.jpg",
    name: "John MacDonald",
    age: 50,
    email: "john.macdonald@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:8"
}
```

```
{
  identity: 9,
  labels: ["Customer"],
  properties: {
    password: "a2dffdrecde",
    gender: "Male",
    image_path: "static/media/profile_pictures/jason_parsons.jpg",
    name: "Jason Parsons",
    age: 45,
    email: "jason.parsons@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:9"
}
```

```
{
  identity: 10,
  labels: ["Customer"],
  properties: {
    password: "a2recde",
    gender: "Male",
    image_path: "static/media/profile_pictures/ian_marshall.jpg",
    name: "Ian Marshall",
    age: 38,
    email: "ian.marshall@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:10"
}
```

```
{
  identity: 11,
  labels: ["Customer"],
  properties: {
    password: "a2recde",
    gender: "Female",
    image_path: "static/media/profile_pictures/emily_hunter.j
    name: "Emily Hunter",
    age: 28,
    email: "emily.hunter@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:11"
}
```

```
{
  identity: 12,
  labels: ["Customer"],
  properties: {
    password: "a2rec88de",
    gender: "Female",
    image_path: "static/media/profile_pictures/sue_graham.jpg",
    name: "Sue Graham",
    age: 26,
    email: "sue.graham@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12"
}
```

```
{
  identity: 13,
  labels: ["Customer"],
  properties: {
    password: "a2rWW8de",
    gender: "Female",
    image_path: "static/media/profile_pictures/michelle_watson.jpg",
    name: "Michelle Watson",
    age: 48,
    email: "michelle.watson@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:13"
}
```

```
{
  identity: 40,
  labels: ["Customer"],
  properties: {
    password: "12345678",
    gender: "male",
    name: "Samyak",
    age: "23",
    email: "samyak.kumbhalwar@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:40"
}
```

```
{
  identity: 41,
  labels: ["Customer"],
  properties: {
    password: "password",
    gender: "female",
    image_path: "static/media/profile_pictures/Numériser2.jpeg",
    name: "Lilou Sicard-Noel",
    email: "lilousn@gmail.com",
    age: "21"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:41"
```

```
{
  identity: 51,
  labels: ["Customer"],
  properties: {
    password: "hello",
    gender: "Male",
    image_path: "static/media/profile_pictures/oscard_donald.jpg",
    name: "Oscard Donald",
    age: 22,
    email: "oscard.donald@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:51"
}
```

```
{
  identity: 49,
  labels: ["Customer"],
  properties: {
    password: "hello",
    gender: "Female",
    image_path: "static/media/profile_pictures/mariana_dona.jpg",
    name: "Mariana Dona",
    age: 22,
    email: "mariana.dona@gmail.com"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:49"
}
```


Restaurant Nodes


```
{
  identity: 33,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest15.jpg",
    name: "La Banquise",
    type: "Poutinerie"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:33"
}
```

```
{
  identity: 32,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest7.jpg",
    name: "Au Pied de Cochon",
    type: "Canadian"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:32"
}
```

```
{
  identity: 31,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest5.jpg",
    name: "La Fabrique de Bagel",
    type: "Bagel"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:31"
}
```

```
{
  identity: 30,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest13.jpg",
    name: "Toro",
    type: "Spanish"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:30"
}
```

```
{
  identity: 29,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest2.jpg",
    name: "Atlantic Fish",
    type: "Seafood"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:29"
}
```

```
{
  identity: 28,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/shakeShack.jpg",
    name: "Boston Shake Shack",
    type: "Burger"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:28"
}
```

```
{
  identity: 27,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/shakeShack.jpg",
    name: "NY Shake Shack",
    type: "Burger"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:27"
}
```

```
{
  identity: 26,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest14.jpg",
    name: "Gramercy Tavern",
    type: "American"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:26"
}
```

```
{
  identity: 25,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest1.jpg",
    name: "Bibble & Sip",
    type: "Bakery"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:25"
}
```

```
{
  identity: 24,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest3.jpg",
    name: "Espetus Churrascaria",
    type: "Brazilian Steakhouses"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:24"
}
```

```
{
  identity: 23,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest12.jpg",
    name: "The Codmother Fish & Chips",
    type: "Seafood"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:23"
}
```

```
{
  identity: 22,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest4.jpg",
    name: "Chapeau",
    type: "French"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:22"
}
```

```
{
  identity: 21,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest6.jpg",
    name: "The Breakfast Club",
    type: "American"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:21"
}
```

```
{
  identity: 20,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest8.jpg",
    name: "Cha Cha Sushi",
    type: "Sushi"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:20"
}
```

```
{
  identity: 19,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest11.jpg",
    name: "The Table",
    type: "American"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:19"
}
```

```
{
  identity: 1,
  labels: ["Restaurant"],
  properties: {
    image_path: "static/media/restaurant_pictures/rest9.jpg",
    name: "Ettan",
    type: "Indian"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:1"
}
```

Rating Nodes


```
{
  identity: 60,
  labels: ["Rating"],
  properties: {
    Comment: "Nice but dark",
    Score: "4"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:60"
}
```

```
{
  identity: 59,
  labels: ["Rating"],
  properties: {
    Comment: "Shake Shack is the best",
    Score: "4"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:59"
}
```

```
{
  identity: 58,
  labels: ["Rating"],
  properties: {
    Comment: "First time trying Indian food. Is it always this good?",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:58"
}
```

```
{
  identity: 57,
  labels: ["Rating"],
  properties: {
    Comment: "I love Bagel",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:57"
}
```

```
{
  identity: 56,
  labels: ["Rating"],
  properties: {
    Comment: "The coffee is so good",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:56"
}
```

```
{
  identity: 55,
  labels: ["Rating"],
  properties: {
    Comment: "Nice, but I have seen better",
    Score: "3"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:55"
}
```

```
{
  identity: 54,
  labels: ["Rating"],
  properties: {
    Comment: "It was a nice experience!",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:54"
}
```

```
{
  identity: 50,
  labels: ["Rating"],
  properties: {
    Comment: "loved the juice.",
    Score: "3"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:50"
}
```

```
{
  identity: 48,
  labels: ["Rating"],
  properties: {
    Comment: "My favorite restaurant",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:48"
}
```

```
{
  identity: 47,
  labels: ["Rating"],
  properties: {
    Comment: "Its okay",
    Score: "4"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:47"
}
```

```
{
  identity: 46,
  labels: ["Rating"],
  properties: {
    Comment: "I just love the Shake Shack",
    Score: "5"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:46"
}
```

```
{
  identity: 44,
  labels: ["Rating"],
  properties: {
    Comment: "",
    Score: "4"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:44"
}
```

```
{
  identity: 43,
  labels: ["Rating"],
  properties: {
    Comment: "Did not like it",
    Score: "2"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:43"
}
```

```
{
  identity: 42,
  labels: ["Rating"],
  properties: {
    score: "4",
    Comment: "It is really nice"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:42"
}
```

```
{
  identity: 7,
  labels: ["Rating"],
  properties: {
    Comment: "I loved the food",
    Score: 5
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:7"
}
```


City Nodes


```
{
  identity: 2,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/paloAlto.jpg",
    name: "Palo Alto",
    state: "California"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:2"
}
```

```
{
  identity: 67,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/houston.jpg",
    name: "Houston",
    state: "Texas"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:67"
}
```

```
{
  identity: 66,
  labels: ["City"],
  properties: {
    country: "Canada",
    image_path: "static/media/city_pictures/vancouver.jpg",
    name: "Vancouver",
    state: "British Columbia"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:66"
}
```

```
{
  identity: 65,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/orlando.jpg",
    name: "Orlando",
    state: "Florida"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:65"
}
```

```
{
  identity: 64,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/miami.jpg",
    name: "Miami",
    state: "Florida"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:64"
}
```

```
{
  identity: 63,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/new_orleans.jpg",
    name: "New Orleans",
    state: "Louisiana"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:63"
}
```

```
{
  identity: 62,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/arizona_city.jpg",
    name: "Phoenix",
    state: "Arizona"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:62"
}
```

```
{
  identity: 61,
  labels: ["City"],
  properties: {
    country: "Canada",
    image_path: "static/media/city_pictures/quebec_city.jpg",
    name: "Quebec City",
    state: "Quebec"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:61"
}
```

```
{
  identity: 53,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/portland.jpg",
    name: "Portland",
    state: "Oregon"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:53"
}
```

```
{
  identity: 52,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/seattle.jpg",
    name: "Seattle",
    state: "Washington"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:52"
}
```

```
{
  identity: 18,
  labels: ["City"],
  properties: {
    country: "Canada",
    image_path: "static/media/city_pictures/montreal.jpg",
    name: "Montreal",
    state: "Quebec"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:18"
}
```

```
{
  identity: 17,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/boston.jpg"
    name: "Boston",
    state: "Massachusetts"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:17"
}
```

```
{
  identity: 16,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/newYork.jpg",
    name: "New York City",
    state: "New York"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:16"
}
```

```
{
  identity: 14,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/sanJose.jpg",
    name: "San Jose",
    state: "California"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:14"
}
```

```
{
  identity: 15,
  labels: ["City"],
  properties: {
    country: "United States",
    image_path: "static/media/city_pictures/sanFrancisco.jpg",
    name: "San Francisco",
    state: "California"
  },
  elementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:15"
}
```

Friends Relationship

"Jason Parsons"	"Jesse"	{
		identity: 102,
		start: 9,
		end: 45,
		type: "Friends",
		properties: {},
		elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:102",
		startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:9",
		endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:45"
		}

"Jesse"	"Jason Parsons"	{
		identity: 101,
		start: 45,
		end: 9,
		type: "Friends",
		properties: {},
		elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:101",
		startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:45"
		endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:9"
		}

"Maria Doe"	"Lilou Sicard-Noel"	{
		identity: 100,
		start: 4,
		end: 41,
		type: "Friends",
		properties: {},
		elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:100",
		startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4",
		endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:41"
		}

"Lilou Sicard-Noel"	"Maria Doe"	{
		identity: 99,
		start: 41,
		end: 4,
		type: "Friends",
		properties: {},
		elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:99",
		startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:41"
		endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4"
		}

"Sue Graham"	"Maria Doe"	{
		identity: 98,
		start: 12,
		end: 4,
		type: "Friends",
		properties: {},
		elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:98",
		startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12"
		endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4"
		}

"Maria Doe"	"Sue Graham"	{ identity: 97, start: 4, end: 12, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:97", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" }
"Sue Graham"	"Nathan Davidson"	{ identity: 15, start: 12, end: 3, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:15", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:3" }
"John Way"	"Eric Graham"	{ identity: 14, start: 5, end: 0, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:14", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:5", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:0" }
"Sue Graham"	"John MacDonald"	{ identity: 13, start: 12, end: 8, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:13", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:8" }
"Maria Doe"	"Nathan Davidson"	{ identity: 12, start: 4, end: 3, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:12", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:3" }

"Sue Graham"	"Eric Graham"	{ identity: 11, start: 12, end: 0, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:11", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:0" }
"Michelle Watson"	"Sue Graham"	{ identity: 10, start: 13, end: 12, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:10", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:13" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" }
"Ian Marshall"	"Sue Graham"	{ identity: 9, start: 10, end: 12, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:9", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:10" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:12" }
"Jason Parsons"	"John MacDonald"	{ identity: 8, start: 9, end: 8, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:8", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:9", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:8" }
"Ian Marshall"	"Emily Hunter"	{ identity: 7, start: 10, end: 11, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:7", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:10" endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:11" }

"Jason Parsons"	"Maria Doe"	{ identity: 6, start: 9, end: 4, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:6", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:9", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4" }
"John Way"	"Maria Doe"	{ identity: 0, start: 5, end: 4, type: "Friends", properties: {}, elementId: "5:dbacc510-7974-4310-a39b-4e98f6452132:0", startNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:5", endNodeElementId: "4:dbacc510-7974-4310-a39b-4e98f6452132:4" }

Likes Relationship

"Michelle Watson"	"NY Shake Shack"	"Likes"
"Michelle Watson"	"Boston Shake Shack"	"Likes"
"Michelle Watson"	"Atlantic Fish"	"Likes"
"Lilou Sicard-Noel"	"Ettan"	"Likes"
"Jason Parsons"	"Cha Cha Sushi"	"Likes"
"Jason Parsons"	"The Breakfast Club"	"Likes"
"Ian Marshall"	"The Table"	"Likes"
"Ian Marshall"	"Chapeau"	"Likes"
"Ian Marshall"	"Espetus Churrascaria"	"Likes"
"Emily Hunter"	"Espetus Churrascaria"	"Likes"
"Maria Doe"	"La Banquise"	"Likes"
"Maria Doe"	"The Breakfast Club"	"Likes"
"John Way"	"The Codmother Fish & Chips"	"Likes"
"John Way"	"Espetus Churrascaria"	"Likes"
"John Way"	"Gramercy Tavern"	"Likes"
"Emily Hunter"	"The Codmother Fish & Chips"	"Likes"
"Emily Hunter"	"Bibble & Sip"	"Likes"
"Sue Graham"	"Bibble & Sip"	"Likes"
"Sue Graham"	"Gramercy Tavern"	"Likes"
"Sue Graham"	"NY Shake Shack"	"Likes"
"Michelle	"NY Shake	"Likes"
"John MacDonald"	"Gramercy Tavern"	"Likes"
"John MacDonald"	"NY Shake Shack"	"Likes"
"John MacDonald"	"Boston Shake Shack"	"Likes"
"John MacDonald"	"Atlantic Fish"	"Likes"
"Jason Parsons"	"The Table"	"Likes"
"John Way"	"Ettan"	"Likes"
"Nathan Davidson"	"Cha Cha Sushi"	"Likes"
"Nathan Davidson"	"Espetus Churrascaria"	"Likes"
"Nathan Davidson"	"Atlantic Fish"	"Likes"
"Maria Doe"	"Atlantic Fish"	"Likes"
"Maria Doe"	"Toro"	"Likes"

Reside Relationship

"Jesse"	"Palo Alto"	"Reside"
"Sue Graham"	"Montreal"	"Reside"
"Michelle Watson"	"Montreal"	"Reside"
"Samyak"	"San Jose"	"Reside"
"Lilou Sicard-Noel"	"San Jose"	"Reside"
"Emily Hunter"	"San Jose"	"Reside"
"Oscard Donald"	"Seattle"	"Reside"
"Mariana	"Portland"	"Reside"
"Eric Graham"	"Palo Alto"	"Reside"
"Nathan Davidson"	"Palo Alto"	"Reside"
"Maria Doe"	"Palo Alto"	"Reside"
"John Way"	"San Francisco"	"Reside"
"John MacDonald"	"San Francisco"	"Reside"
"Jason Parsons"	"San Francisco"	"Reside"
"Ian Marshall"	"New York City"	"Reside"

Location Relationship

"Bibble & Sip"	"New York City"	"Location"
"Gramercy Tavern"	"New York City"	"Location"
"NY Shake Shack"	"New York City"	"Location"
"Boston Shake Shack"	"Boston"	"Location"
"Atlantic Fish"	"Boston"	"Location"
"Toro"	"Boston"	"Location"
"La Fabrique de Bagel"	"Montreal"	"Location"
"Au Pied de Cochon"	"Montreal"	"Location"
"La Banquise"	"Montreal"	"Location"
"Ettan"	"Palo Alto"	"Location"
"The Table"	"San Jose"	"Location"
"Cha Cha Sushi"	"San Jose"	"Location"
"The Breakfast Club"	"San Jose"	"Location"
"Chapeau"	"San Francisco"	"Location"
"The Codmother Fish & Chips"	"San Francisco"	"Location"
"Espetus Churrascaria"	"San Francisco"	"Location"

Made and Review Relationship

"Mariana Dona"	"Made"	"First time trying Indian food. Is it always this good?"	"5"	"Review"	"Ettan"
"Mariana Dona"	"Made"	"Shake Shack is the best"	"4"	"Review"	"Boston Shake Shack"
"Mariana Dona"	"Made"	"Nice but dark"	"4"	"Review"	"Gramercy Tavern"
"Jesse"	"Made"	"loved the juice."	"3"	"Review"	"Espetus Churrascaria"
"Oscard Donald"	"Made"	"It was a nice experience!"	"5"	"Review"	"Cha Cha Sushi"
"Oscard Donald"	"Made"	"Nice, but I have seen better"	"3"	"Review"	"The Table"
"Oscard Donald"	"Made"	"The coffee is so good"	"5"	"Review"	"Bibble & Sip"
"Oscard Donald"	"Made"	"I love Bagel"	"5"	"Review"	"La Fabrique de Bagel"
"Maria Doe"	"Made"	"I loved the food"	5	"Review"	"Ettan"
"Sue Graham"	"Made"	"I just love the Shake Shack"	"5"	"Review"	"NY Shake Shack"
"Jesse"	"Made"	"Its okay"	"4"	"Review"	"The Breakfast Club"
"Sue Graham"	"Made"	"Did not like it"	"2"	"Review"	"The Table"
"Sue Graham"	"Made"	""	"4"	"Review"	"Cha Cha Sushi"
"Sue Graham"	"Made"	"It is really nice"	*null*	"Review"	"Gramercy Tavern"
"Lilou Sicard-Noel"	"Made"	"My favorite restaurant"	"5"	"Review"	"Ettan"