Final Project Report

CIS550: Database & Info System

By Alicia Teo, Christina Chen, Fred Chang, Lucy Kuo

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

I. Introduction

Abstract
When choosing a webapp to create, we thought about our own hobbies and interests. As a group of avid travelers, with one member being a Yelp Elite restaurant reviewer and another being a credit card points and miles hobbyist, we thought about all the time we spend planning where to go, what to do, and where to stay. We were frustrated with having to visit numerous websites (Yelp, Hotel websites, Google places to see and things to do) and having to compile for ourselves an overview of what we can do in a new city. We decided to combine our interests and create a useful webapp for others who like to travel, sightsee and eat. Reminiscent of Julia Roberts' famous soul-searching travels in the movie, "Eat Pray Love," we have named our webapp **EatStayLove**.

Introduction and project goals

Target Problem**:**
Today, if one were to plan a trip, including what to see, where to stay, and what to eat, it involves cross referencing multiple sources like Google Maps, hotel websites, Expedia, Yelp, Foursquare, etc.

Project Goals:
We created a webapp, called **EatStayLove**, which integrates everything by becoming the **one-stop-shop** for helping you find all the best hotels and restaurants based on the sights you wish to see.
Our project goals include the following:

- Provide a user with a complete view of hotel, dining, and attractions in a specified city.
- Display hotels and restaurants customized to the user's cuisine and ratings preferences.
- Recommend the top highly rated restaurants to a user in a city.
- Ultimately help users create a travel itinerary based on what to eat, where to stay, and what to see!

Description of App Functionality:
The user chooses a city and landmarks to visit, and the app will provide hotel and dining suggestions that are in the area. The app will then summarize the user's itinerary.

- Landmark selector (user can add to a saved list all places they wish to visit)
- Hotel recommendation system (users will be shown hotel recommendations)
- Restaurant recommendation system (users will be shown nearby restaurant recommendations with curated restaurant reviews)
- Trip summary (Users will see the landmarks they have chosen, the hotel, and restaurants they have chosen)

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

II. Architecture

List of technologies:

- **Front End:** AngularJS, HTML, and CSS
- **Middleware:** NodeJS
- **Database:** We used an oracle database hosted on AWS RDS

APIs used:

- **Foursquare API:**
  We used the Foursquare API to get information on city landmarks.

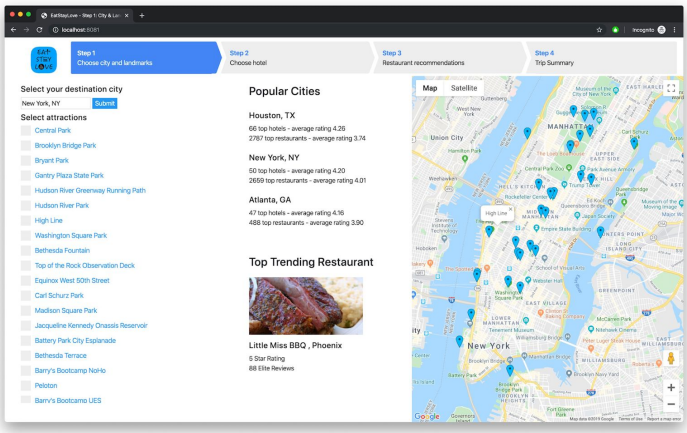
- **Google Maps Javascript API:**
  We used Google Maps JS API to show custom maps on every page of our app. Once users select landmarks, hotels, and restaurants, we pass the address and latitude / longitude to Google Maps to display on a map on our app. (*Link:* https://developers.google.com/maps/documentation/javascript/get-api-key)

Description of system architecture/application:

The architecture of our webapp is essentially divided into two components: client and server. The client side is the web user and the server side is AWS. HTTP GET request/response is the communication between the client and server sides.

Below are descriptions of our web pages with information about what the respective pages accomplish:

Welcome Page:

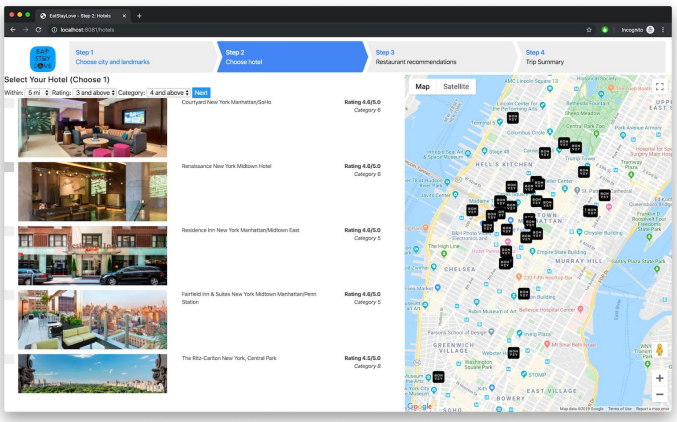


When a user comes to our webapp, they will immediately get a recommendation of the top 3 cities with the most highly rated restaurants and hotels having a rating of 4 or higher out of 5.

After seeing these 3 popular city recommendations, the user is given the option to select their own destination city. When they click next, they get a list of the top rated attractions and a map of the landmarks.

In the city chosen by the user, he or she can then select one or more interesting attractions they want to see during their visit.

Hotels Page:

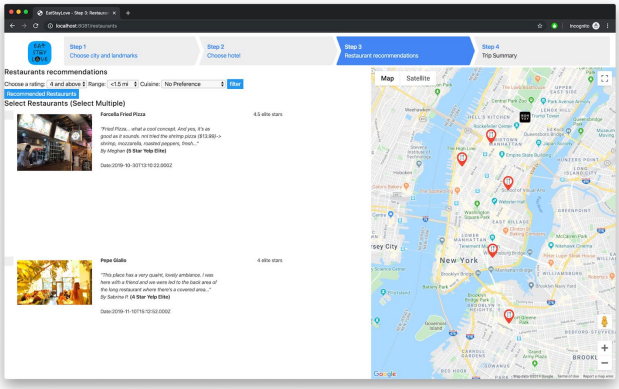


After choosing the specific landmarks a user wants to see, he or she can choose a hotel

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

based on how many miles the hotels are from the attractions. Based on the number of miles selected by the user, we show hotels that are located within a radius of that distance from any attraction selected by the user. The hotels can be ranked by ratings and categories.

Restaurants Page:

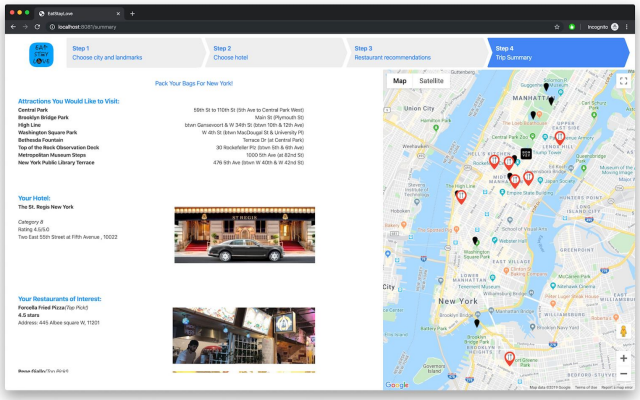


A user is able to see restaurants in two ways: First, users are able to pick restaurants at or above a certain Yelp rating.

Second, a user can see recommended restaurants that have the highest rating and the most number of reviews written by Yelp Elite members in the city the user chose.

For each restaurant, we show photos from Yelp and the restaurant locations are displayed on a map so that users can easily see where they are located and what the food and restaurant looks like.

Summary Page:



The final page a user sees acts as a full itinerary summary that is easily printable for the user to embark on their trip.

All the landmarks, hotels, and restaurants chosen by the user are displayed on a map.

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

III. Data

Below are the descriptions and explanations for how we have used the underlying data sets in our app:

Yelp Restaurants & Reviews

Description:
Our restaurant data comes from a downloadable dataset provided by the Yelp Dataset Challenge (https://www.yelp.com/dataset/challenge).

We also obtained additional restaurants data by calling the Yelp API (https://www.yelp.com/developers/documentation/v3/get_started)

Explanation for our use:
Using this data, we show our users recommended restaurants near their chosen hotel.

Relevant summary statistics:
- 357,204 restaurants
- 630,636 photos
- 1,024,042 restaurant attributes (e.g. 'Breakfast')
- 2,366,336 restaurant reviews
- 1,891,388 yelp reviewers

Marriott Hotels:

Description:
We scraped hotel data from Marriott's website. This data lists all hotels, price category, hotel address, and link to hotel image for hotels in the United States.

Link: https://www.marriott.com/hotel-search.mi

Relevant summary statistics:
- 5,274 hotels

Explanation for our use:
We use Marriott hotel data for queries that run in combination with the other data sets. Some examples of our use of the data are:
- Show recommended cities with the most highest rated hotels (4 or above) and the most highest rated restaurants (4 or above).
- Return cities where there are at least 5 hotels and 50 restaurants.
- Show hotels' locations on a map.

Foursquare Landmarks:

We got information on city landmarks from Foursquare via API requests. Once the user selects a city, we send an API request to Foursquare to return the list of landmarks located within the city.

Link:
https://developer.foursquare.com/docs/api

Explanation for use:
- Allow users to see the attractions in the city they selected.
- Display the attractions on a Google Map near the hotels and restaurants selected by the user.

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

IV. Database

Data Ingestion:

Yelp Restaurants & Reviews:
Data from the Yelp Dataset challenge was provided in JSON format. We first converted JSON into csv format for better readability. Once we determined what data we needed, we then parsed the csv into DDL statements stored in a .sql file. We then uploaded the DDL statements into our AWS RDS database.

To get data from the Yelp API, we called the API and stored all data as csv, and then parsed each csv line into DDL statements.

Marriott Hotels
We scraped data from Marriott into a csv file. We used python to do some data clean up, and then converted the csv into DDL.

Relational Schema & Entity Resolution:

Original Data Structure Pre-Entity Resolution
Structure of the original data when retrieved from sources:

Hotel(name, category, address, analytics, logo, img)
Restaurant(business_id, name, address, city, state, latitude, longitude, stars, review_count, categories)
Review(review_id, user_id, business_id, useful, funny, cool, text, date)
Photo(caption, photo_id, business_id, label)

Relational schema after entity resolution, including normal forms and justifications:

City(city_name, state)

Num Instances: 7,163
Normal Form: BCNF
Justification: city_name and state together form the primary key and there are no non-key attributes.

Hotel(hotel_id, hotel_name, city_name, state, zipcode, latitude, longitude, address, full_address, category, num_reviews, rating, img)
city_name, state fk linking to city(city_name, state)

Num Instances: 5,274
Normal Form: 2NF
Justification:
hotel_id is the key.
FDs are:
hotel_id -> city, state, address
hotel_id -> zipcode
city, state, address -> zipcode

However, for our app, we frequently need to display all the address information together (i.e. city, state, address, zipcode, latitude, longitude), and to avoid doing many additional joins, it makes sense to keep all the address information together in one table instead of breaking it out into multiple tables to achieve 3NF.

Restaurant(restaurant_id, is_restaurant, restaurant_name, latitude, longitude, address, city_name, state, zipcode, categories, stars, review_count)
city_name, state fk linking to city(city_name, state)

Num Instances: 357,204

Normal Form: 2NF
Justification:
restaurant_id is the key.
FDs are:
restaurant_id -> city, state, address
restaurant_id -> zipcode
city, state, address -> zipcode

Justification: We frequently want to display restaurant address information together, so it makes sense to keep all the address information together.

Restaurant_attributes(restaurant_id, attribute)
restaurnt_id fk linking to restaurant(restaurant_id)

Num Instances: 1,024,042
Normal Form: BCNF
Justification: restaurant_id and attribute together form the primary key.

Restaurant_photos(photo_id, restaurant_id, caption, label)
restaurant_id fk linking to restaurant(restaurant_id)

Num Instances: 630,636
Normal Form: BCNF
Justification:
photo_id is the key for the table.

FDs are:
photo_id -> restaurant_id
photo_id -> caption
photo_id -> label
Every left side of an FD is a key for the table.

Restaurant_reviews(restaurant_id, review_id, reviewer_id, stars, review_text, review_date)
restaurant_id fk linking to restaurant(restaurant_id)
reviewer_id fk linking to yelp_reviewers(user_id)

Num Instances: 2,366,336
Normal Form: BCNF
Justification:

Keys: review_id
FDs:
review_id -> reviewer_id
review_id -> restaurant_id
review_id -> review_date
review_id -> review_text
review_id -> stars

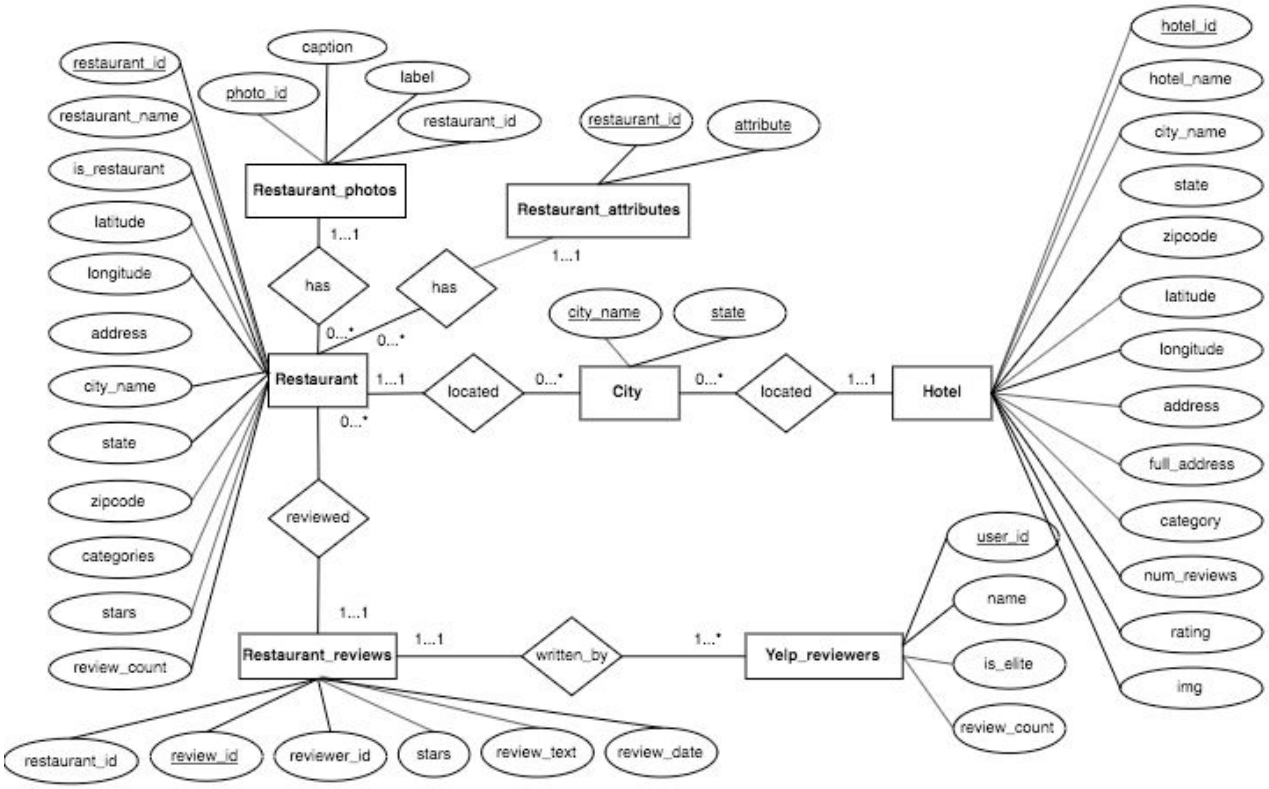
Every left hand side of the all functional dependencies is a key for the relation. We considered whether the number of stars or review date would be functionally determined by the review_text, but we checked our data and some reviews have the same text but different star ratings.

Yelp_reviewers(user_id, name, is_elite, review_count)

Num Instances: 1,891,388
Normal Form: BCNF
Justification:

Keys: user_id
FDs:
user_id -> name
user_id -> is_elite
user_id -> review_count
Every left hand side of the all functional dependencies is a key for the relation. We considered whether is_elite would be functionally dependent on name, but we are assuming correctly that multiple Yelp reviewers can have the same name and not all of them will be Elite.

ER diagram:


restaurant_id
restaurant_name
is_restaurant
latitude
longitude
address
city_name
state
zipcode
categories
stars
review_count
caption
photo_id
label
restaurant_id
Restaurant_photos
restaurant_id
attribute
Restaurant_attributes
1...1
has
has
0...*
0...*
Restaurant
1...1
located
0...*
City
city_name
state
0...*
located
1...1
Hotel
0...*
reviewed
1...1
Restaurant_reviews
1...1
written_by
1...*
Yelp_reviewers
hotel_id
hotel_name
city_name
state
zipcode
latitude
longitude
address
full_address
category
num_reviews
rating
img
user_id
name
is_elite
review_count
restaurant_id
review_id
reviewer_id
stars
review_text
review_date

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

V. Queries

Most complex queries are: Queries 1, 2, and 6

1) Recommends 5 top restaurants in a city to users on the 'Restaurants' page.

Show 5 restaurants in {city} with the highest overall rating (average rating by users + average rating by Yelp Elite)

This query is one of the most complex.

Result schema:
(restaurant_id, restaurant_name, stars, elite_stars, review_text, review_date, reviewer, reviewer_stars, photo_id, address, zipcode, latitude, longitude)

```
/** find the avg yelp elite rating **/
WITH t AS (
   SELECT r.restaurant_id, avg(r.stars)
elite_stars
   FROM restaurant r
   JOIN restaurant_reviews rr ON
R.restaurant_id = rr.restaurant_id
   JOIN yelp_reviewers y ON
rr.reviewer_id = y.user_id
   WHERE r.is_restaurant = 1 AND
y.is_elite = 1
   AND r.city_name = 'New York' and
r.state = 'NY'
   GROUP by r.restaurant_id
),
/** Get latest yelp elite review **/
rec_rev AS (
   SELECT rr.restaurant_id,
rr.review_text, rr.stars
reviewer_stars, rr.review_date, y.name
   FROM restaurant_reviews rr
   INNER JOIN (
            SELECT rr.restaurant_id,
MAX(rr.review_date) review_date
            FROM restaurant_reviews rr
            JOIN yelp_reviewers y on
rr.reviewer_id = y.user_id
            WHERE y.is_elite = 1
            GROUP BY rr.restaurant_id
       ) x
   ON rr.restaurant_id =
x.restaurant_id AND rr.review_date =
x.review_date
   JOIN yelp_reviewers y on
rr.reviewer_id = y.user_id
)
/** get overall rating , limit to 5
restaurants **/
SELECT t.restaurant_id,
r.restaurant_name,
r.stars, t.elite_stars,
rec_rev.review_text,
rec_rev.review_date, rec_rev.name
reviewer,
rec_rev.reviewer_stars, x5.photo_id,
r.address, r.zipcode, r.latitude,
r.longitude
FROM t
JOIN rec_rev on t.restaurant_id =
rec_rev.restaurant_id
JOIN restaurant r ON t.restaurant_id =
r.restaurant_id
/** Get photo for restaurant **/
INNER JOIN (
   SELECT rp.restaurant_id,
MAX(rp.photo_id) photo_id
   FROM restaurant_photos rp
   WHERE rp.photo_id is not null
   GROUP BY rp.restaurant_id
) x5
ON x5.restaurant_id = t.restaurant_id
/* Set city name and State*/
WHERE r.city_name = 'New York' and
r.state = 'NY'
AND r.is_restaurant = 1
AND rownum <= 5
ORDER BY (r.stars + t.elite_stars) DESC
;
```

2) On the homepage, show recommended cities to users before users pick a city to visit.

Show 3 cities with the most number of hotels with ratings of 4 or higher. For each of the 3 cities, show the number of restaurants with ratings of 4 or higher, and the average restaurant rating in that city.

This query is one of the most complex.

Result schema:

(city_name, state, num_hr_hotels, avg_hotel_rating, num_restaurants, avg_rest_rating)

```
/** Get top 3 highly rated cities **/
WITH highlyRated AS (
SELECT city_name, state, num_hr_hotels
FROM (
   SELECT city_name, state, count(*) as
num_hr_hotels
   FROM hotel
   WHERE rating >= 4
   GROUP by city_name, state
   ORDER BY num_hr_hotels DESC)
   WHERE ROWNUM <= 5)
SELECT hr.city_name, hr.state,
hr.num_hr_hotels, x3.avg_hotel_rating,
x1.num_highly_restaurants,
x4.avg_rest_rating
FROM highlyRated hr
JOIN
/** Get avg hotel rating for the city
**/
(SELECT city_name, state, avg(rating)
avg_hotel_rating
FROM hotel
GROUP BY city_name, state) x3
ON hr.city_name = x3.city_name AND
hr.state = x3.state
/** Get num of highly rated restaurants
**/
JOIN
(
   SELECT r.city_name, r.state,
count(*) num_highly_restaurants
   FROM restaurant r
   WHERE r.stars >= 4
   GROUP BY r.city_name, r.state
) x1
ON hr.city_name = x1.city_name AND
hr.state = x1.state
/** Find avg restaurant rating for the
city **/
JOIN (
   SELECT city_name, state, avg(stars)
avg_rest_rating
   FROM restaurant r
   GROUP BY r.city_name, r.state
) x4
ON hr.city_name = x4.city_name AND
hr.state = x4.state;
```

3) On the homepage, provide a dropdown list of cities for users to choose to visit.

Provides a list of cities for which there at least 5 hotels and 50 restaurants in each city in our dataset.

Result schema: (city_name, state, num_hotels, num_restaurants)

```
WITH X AS (
SELECT h1.*, r1.num_restaurants
FROM
(
SELECT h.city_name, h.state, count(*)
num_hotels
FROM hotel h
group by h.city_name, h.state
HAVING count(*) > 0
ORDER BY count(*) DESC
) h1
JOIN
(
   select r.city_name, r.state,
count(*) num_restaurants
   from restaurant r
   group by r.city_name, r.state) r1
ON h1.city_name = r1.city_name and
r1.state = h1.state)
SELECT X.*
FROM X
WHERE X.num_restaurants > 50 AND
X.num_hotels >= 5
ORDER BY X.num_hotels DESC;
```

4) On the 'Restaurants' page, provide a list of top 20 cuisines, and number of restaurants in each cuisine. Users can filter for restaurants in each cuisine type.

For restaurants in the city, find the top 20 cuisines with the most restaurants
Schema: (attribute, num_restaurants)

```
SELECT *
FROM (
   SELECT ra.attribute,
count(r.restaurant_id) num_restaurants
   FROM restaurant r
   JOIN restaurant_attributes ra
   ON r.restaurant_id =
ra.restaurant_id
   WHERE r.city_name = 'New York' AND
r.state = 'NY' AND r.is_restaurant = 1
   GROUP BY r.city_name, r.state,
ra.attribute
   ORDER BY num_restaurants DESC
```

```
)
WHERE ROWNUM <= 20;
```

5) On the 'Restaurants' page, once the user selects a cuisine type, filter restaurants by cuisine, and show the address, restaurant name, and restaurant photo of the restaurants with that cuisine.

Schema: (restaurant_id, restaurant_name, stars, review_count, latitude, longitude, address, categories, photo_id)

```
SELECT R1.restaurant_id,
R1.restaurant_name,
   R1.stars,
   R1.review_count,
   R1.latitude,
   R1.longitude,
   R1.address,
   R1.categories,
   R2.photo_id
   FROM
        (SELECT r.*
        FROM Restaurant r
        WHERE r.is_restaurant = 1
        AND r.categories like
('%Sandwiches%')
        AND r.stars >= 1
        AND r.review_count > 10
        AND r.state = 'NY'
        ORDER BY r.stars DESC
        ) R1
   JOIN Restaurant_photos R2
   ON R1.restaurant_id =
R2.restaurant_id;
```

6) On the homepage, show a recommended restaurant to users.

Recommends a restaurant which fulfills both criteria below:

- Most number of 5-star yelp elite rating
- Has an average yelp rating of 5

This query is one of the most complex.

```
/** Yelp Elite Rating of 5 **/
SELECT r.restaurant_name, r.city_name,
r.stars, rp.photo_id,
x2.numEliteReviews FROM
(
SELECT r.restaurant_id,
count(r.restaurant_id) as
numEliteReviews
FROM restaurant r
/** restaurants with perfect 5 star
ratings **/
INNER JOIN (
   SELECT r.restaurant_id
   FROM restaurant r
   WHERE r.stars = 5 and
r.is_restaurant = 1
) x on r.restaurant_id =
x.restaurant_id
/** get yelp elite reviews **/
JOIN restaurant_reviews rr ON
r.restaurant_id = rr.restaurant_id
JOIN yelp_reviewers yr on
rr.reviewer_id = yr.user_id
WHERE rr.stars = 5 AND yr.is_elite = 1
GROUP BY r.restaurant_id
ORDER BY numEliteReviews DESC) x2

JOIN restaurant r on
x2.restaurant_id = r.restaurant_id
JOIN restaurant_photos rp on
x2.restaurant_id = rp.restaurant_id
WHERE ROWNUM <= 1;
```

VI. Performance Evaluation

Below is a list of events with recorded timings and explanations for how optimizations have improved timings.

Query numbers refer to the numberings listed in section V.

Query #1:
This query generates 3 restaurants in a specified city, state with the highest overall rating consisting of the sum of the rating and the average Yelp Elite rating.

Steps	**Runtime (seconds)**
Initial Query No indexes on db except on primary keys of each table	30.7145
Indexed hotel.city_name, hotel.state, restaurants.city_name, and restaurants.state	29.3525
Indexed restaurant_photos.restaurant_id, and restaurant_reviews.restaurant_id	0.48
Final runtime	**0.48**

Query #6:
For our front page, in addition to showing the top 3 recommended cities, we wanted to catch users' attention by showing them the must-visit restaurant with a perfect 5/5 Yelp rating and the most number of 5 star reviews written by Yelp Elite members.

Steps	**Runtime (seconds)**
Initial Query	13.32
Created temporary tables that limit the size of joined data sets of restaurants (limited to only those with 5 stars), restaurant reviews (only 5 stars), and Yelp reviewers (only Elite)	11.11
Original query had 3 nested subqueries. Reduced the number of subqueries to 1, and used inner join to replace the other 2 subqueries, reducing the number of full scans needed of the restaurant_reviews table. Used an inner join to reduce the cardinality of the scan of the restaurant table.	10.01
Created index on resturant_reviews (reviewer_id) after noticing that the db was doing a full scan of restaurant_reviews(reviewer_id) in order to join with yelp_reviews(reviewer_id)	6.91
Final runtime	**6.91**

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

Query #5:
This query displays restaurants in a city, state that serve a user-selected cuisine.

Our initial query entailed joining restaurant table with restaurant_attributes table on restaurant_id to get the attribute which matches the cuisine. We optimized the query by getting rid of the join and finding the cuisine string in an array of cuisines in the "categories" attribute of the Restaurant table.

Steps	**Runtime (seconds)**
Initial average runtime (with join operation)	0.767
Final runtime	**0.154**

Query #4:
On the 'Restaurants' page, provide a list of top 20 cuisines, and number of restaurants in each cuisine. Users can filter for restaurants in each cuisine type.

Speed is essential for this query, because users expect to see the full list of cuisines immediately upon clicking the dropdown list.

Steps	**Runtime (seconds)**
Initial average runtime	2.005
Indexed restaurant_attributes(attribute)	1.3
Used count(r.restaurant_id) instead of count(*)	**0.81**

For these and all other queries, we were able to achieve fast runtimes by only joining data sets of information that we need, choosing appropriate join orders with the smaller relation on the outside, and indexing identifying points for large data sets.

VII. Technical Challenges

Below are some of the technical challenges we encountered and explanations for how we successfully overcame them:

Data Persistence Across Multiple Pages:
When we first started, we put all content on the same page to ensure functionality. We then started separating out content into different pages. We needed to move data across pages, especially since we have distance filters to show hotels around a previously chosen city and to show restaurants around a previously chosen hotel. First, we tried to put global variables in the app.js file and see if all the controllers can share the data, but that didn't work. We ultimately decided to save data in local storage (browser). When a user navigates to the next page, we retrieve the saved city and hotel data to do the calculation we need. On the summary page, we simply retrieve all selected data in the browser and display them.

Google Maps Integration:
Google API was a bit more complicated to incorporate so we needed to read a lot of the documentation and test out the API through trial and error and experiment with solutions from online sources. There were many components we figured out such as how to feed markers into Google Maps, how to set the default location and zoom size, how to customize our own logos (e.g. the Marriott logo to show hotel locations and restaurant logos). We achieved this by doing a lot of research including how to use a locally stored image. We also used Photoshop to scale out and customize designs.

Passing Data Back and Forth Through the Stack:
Understanding how the data is requested and passed back through the stack was a major challenge. Although we got a glimpse at the basics of data requests work through HW and recitation, making it work for the project was much more challenging. Minor code changes such as passing multiple parameters to the router resulted in lots of researching and testing of different methods. Furthermore, a core feature of our webapp is to present users the restaurants that are close to the chosen hotel. However, we found it challenging to implement the complex distance calculation algorithm inside SQL, so it resulted in having us handling it in the front end. A benefit of this, we found, is that there is simply one request to the database, and even if a user decides to change distance parameters, the calculation can be done without another query to the database.

Dealing with Images:
We chose to not locally store image files and instead rely on the URL. Since we have photos from multiple sources, we needed the photos to be consistently formatted for our website. We referenced the extra credit section of our homework to help guide us through this.

Getting and Wrangling Yelp Data for the Database:
We initially thought the data from the Yelp Dataset Challenge (https://www.yelp.com/dataset/challenge) would be sufficient. However, when we checked the data in more detail, we realized the Yelp dataset only contained data from 10 cities (and only 7 cities in the US). And many of the cities (e.g. New York, San Francisco, Boston, Seattle) where we had Marriott hotels data were missing in the Yelp dataset.

To overcome this data limitation, we wrote a python script to call the Yelp API to search for restaurants near each of the Marriott hotels in our dataset.

We had to use 13 Yelp API keys to get the data, because each API key could only make 5,000 calls per day, and each API call would only return 50 restaurants (i.e. 1 page of yelp results) at a time. We ultimately added more than 150,000 additional restaurants, 450,000+ photos, 200,000+ reviews, and 200,000+ reviewers to the dataset through getting data via the API.

Cleaning the Yelp data was also challenging because the format returned by the Yelp Dataset Challenge and the Yelp API were formatted differently, which meant we had to create two different scripts to clean the data for our Restaurants table.

Getting and Wrangling Marriott Data for the Database:

We used Data Miner, a Chrome scripting plug-in to scrape the data from the website and export it as a .CSV. A challenge was that Marriott loads their individual hotel data in 40-hotel increments, so we had to configure the plug-in to scrape up until the end of the page. Because the pictures took too long to load, we first scraped the data for every hotel besides the picture on first run and then get the picture on the second scrape. About 40 pages in, Marriott halted our scraping efforts so we waited a few hours and rescraped.

The Marriott dataset did not contain the latitude / longitude of each hotel. So for each hotel, we called the Google Maps Geocoding API to obtain that information.

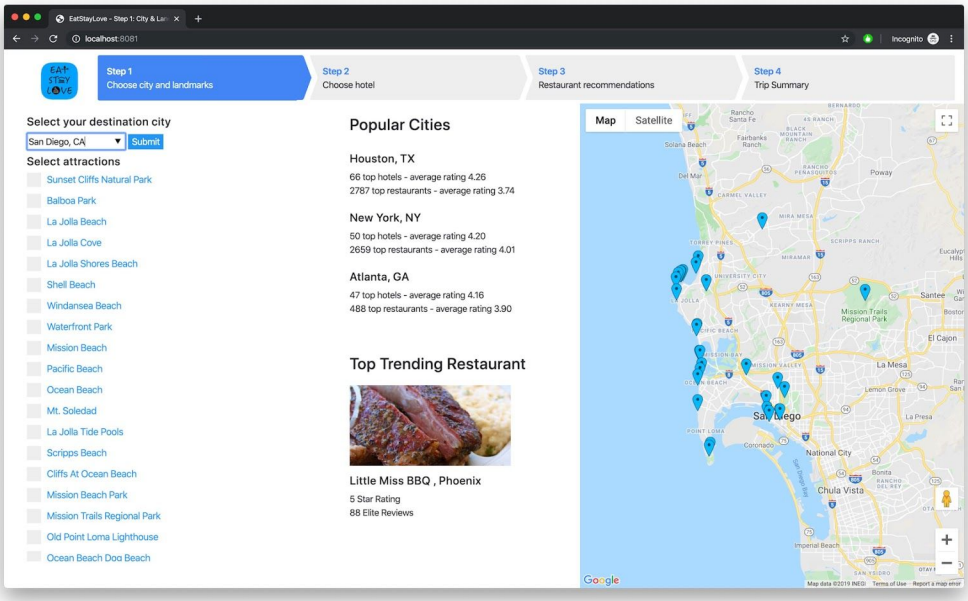
For data cleaning, Marriott data had each hotel's entire address as one string field e.g. 'street address, city, state, zipcode'. We had to parse out the city, state, and zipcode into separate columns.

VIII. Extra Credit

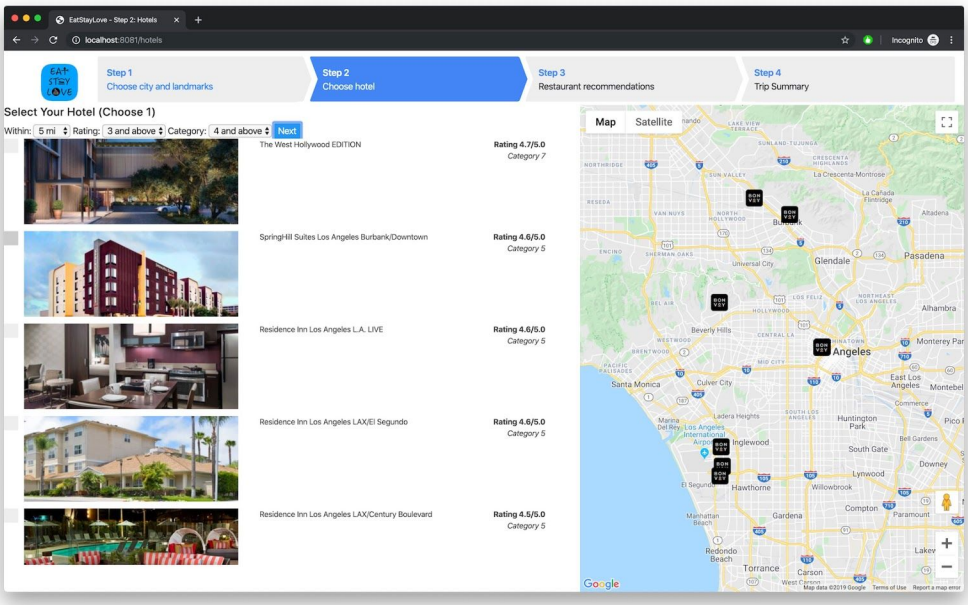
- Used cloud hosting for database - OracleDB hosted on AWS
- Integrated Google Maps Javascript API to show plot of hotels, landmarks, and restaurants on Google Maps
- Integrated various APIs (Yelp, Foursquare)

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo

Additional Web App Screenshots



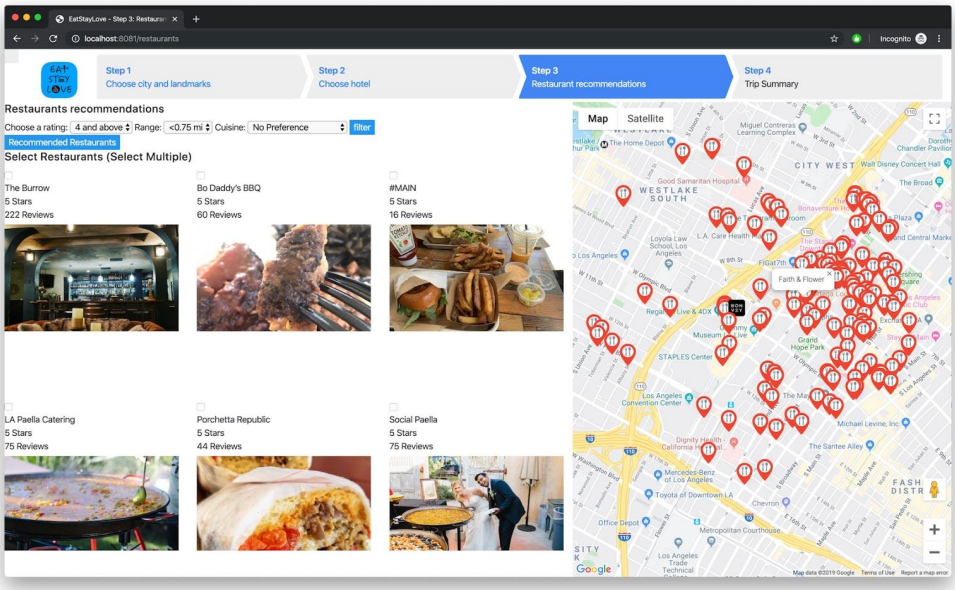
Landmarks in
San Diego, CA



Hotels in
Los Angeles, CA

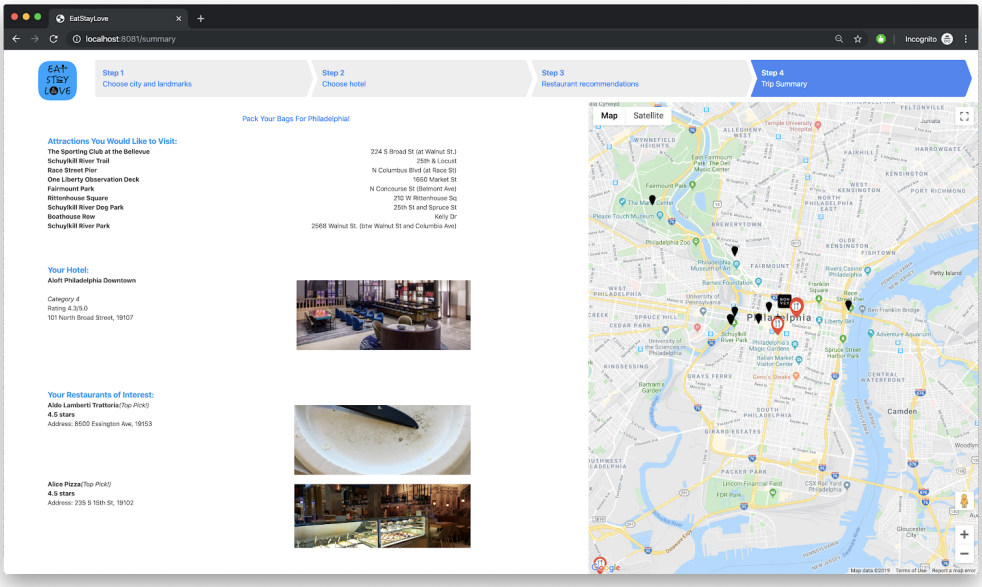
Rating: 3+
Category: 4+

Alicia Teo, Christina Chen
Fred Chang, Lucy Kuo



Recommended Restaurants in Los Angeles, CA

Rating: 4+
Range: <.75m
Cuisine: No Preference



Summary Page for Philadelphia, PA