CS 6365 Final Project Report

Travel Assistant with Parking Planner

Ting-Yu Kang 903443289
Yingqiong Shi 903039969

Abstract

In this project, we designed a web application that is able to auto-generate a travel plan for a user with parking information. It is very useful for a traveler to explore an unfamiliar place without spending lots of time doing homework in advance.

Based on the user's travel preference, the application finds the best attractions and parking lots to form "attraction groups", which are the core feature of our application. After that, we connect the attraction groups and compute the estimated time. Finally, the app returns a complete travel plan back to the user.

The main contribution of our work is to simplify time-consuming travel planning by integrating attraction selection and parking lot finding. Also, our app collects the data from different APIs and use our own algorithm for route planning. By using our application, tourists can easily generate their unique travel plans and start their adventure by one click.

We had deployed our project to the cloud server: https://cs6365final-frontend.herokuapp.com/

Motivation

There are already numerous well-functional websites that can help us find a place to go. However, from our previous experiences, we found it annoying to search again and again if there are many places we want to go. Also, finding a parking lot is nightmare in a city and it might be too far to walk to the attraction. Therefore, we address the problem by designing a web application that integrates attractions finding, parking lot searching with route calculation to give users a comprehensive travel plan. With this application, travelers can start their exploring the city immediately.

Target Users

1. Those who are new to a place and want to take a quick look at the most famous tourist attractions without wasting time searching online.
2. Those who rent a car in a city and worry about finding a parking lot.
3. Those who feel annoying using multiple apps to search for attractions, restaurants, parking lots, and routes.
4. Those who want to get a travel schedule and estimate traffic time in advance instead of frequently finding the next stop in place.

Time-consuming attraction finding and parking lot searching can kill all the joys of traveling. All the concerns above should be perfectly taken cared by a single application so as to let travelers enjoy their trip without caring about any complexities.

Objective

Our goal is to design a web-application that integrates data from different web sources to realize our idea. In our application, there are two searching modes in our application for different needs:

1. **Auto attraction search:** User is open to explore any attractions.
   (1) Based on user-indicated **categories** and current location, automatically find top attractions with parking information.
   (2) Return a route connecting attractions and parking lots with corresponding traveling time.
2. **Specified attraction search:** User wants to indicate specific places to visit.
   (1) Based on user-indicated **attractions** and current location, automatically find parking lots for each attraction with a best traveling order.
   (2) Return a route connecting attractions and parking lots with corresponding traveling time.

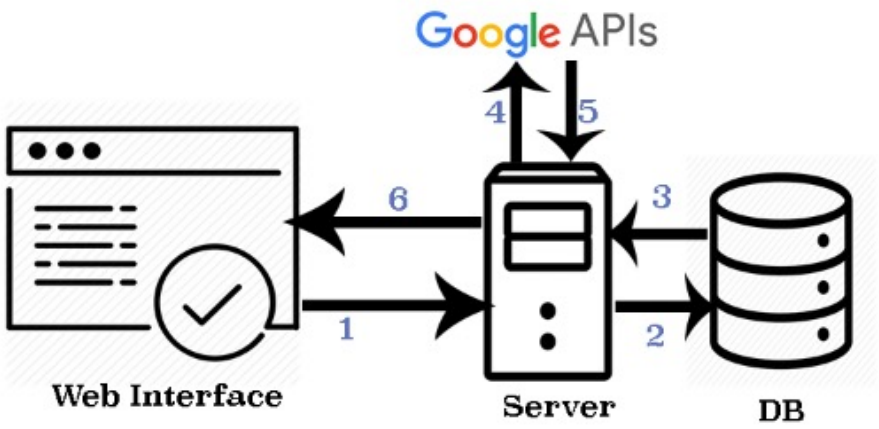
Tools

1. Front-end: React, Material-UI, Bootstrap
2. Back-end: Node, Express framework
3. Database: MongoDB
4. API: Google (Place, Distance Matrix, Maps, Geolocation), Yelp Fusion
5. Web Server: Heroku

Application Architecture

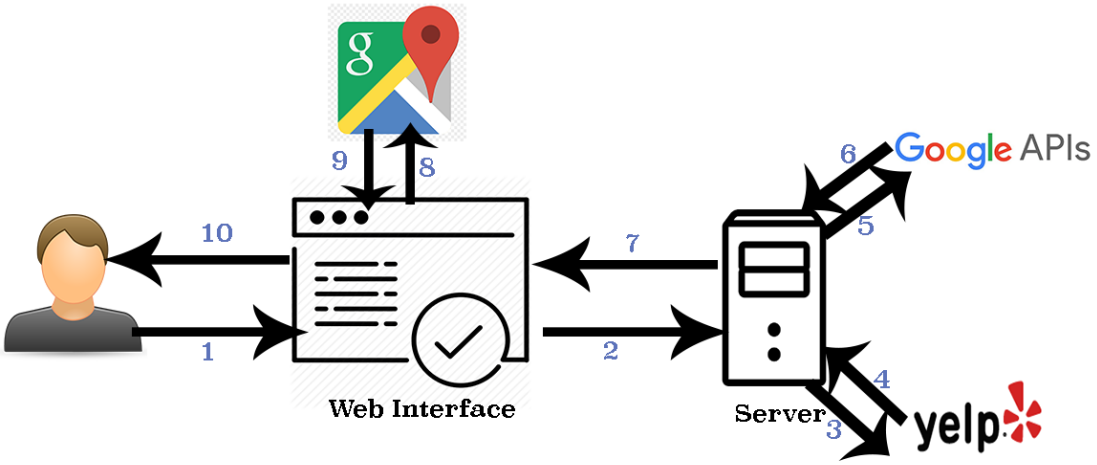
1. Components
   (1) Front-end
      (a) Display **web forms** for users to indicate their preference.
      (b) Send **queries** to backend.
      (c) Show **routes** detail on the map with GoogleMaps API.
   (2) Back-end
      (a) Implement **attraction/parking finding** logic and **route planning** algorithm.
      (b) Based on user preference to get a set of tourist attractions with Yelp API.
      (c) Find **parking lots** for each attraction with Google Place API.
      (d) Calculate **best route** and **travel time** with Google Distance Matrix API.
   (3) Database
      (a) Store manually chosen top attractions
      (b) Store attraction category lists

2. Web Form Preprocess (get choices and list items from database):



(1) Front-end sends an API call to back-end to query the form item contents (Attraction Categories and Famous Attractions) and current location.
(2) Back-end fetches the corresponding data from database.
(3) Database returns the data to server.
(4) Backend-end gets current location from Google Geolocation API.
(5) The API returns the current location.
(6) Server returns the form choice, item contents and current location for front-end to display.

3. For submission and showed calculated routes



(1) User indicates preferred categories or specific travel attractions on web interface.
(2) The front-end sends user preference and current location to the back-end.
(3) The back-end queries Yelp Fusion API to get recommended attractions.
(4) Yelp Fusion API responds with the data.
(5) The back-end queries Google place API to get parking lot information and queries Google Distance Matrix API to get travel time between places.
(6) Google place API and Distance Matrix API responds with the data.

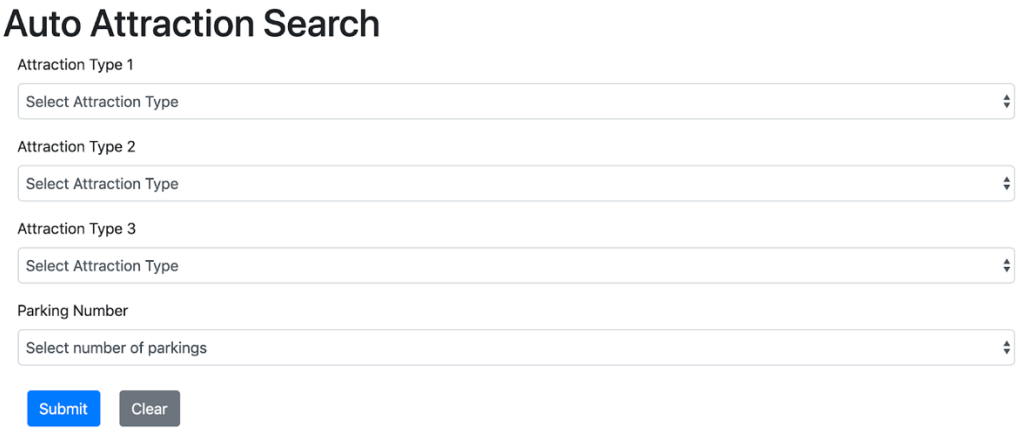
(7) After calculating the best traveling plan, the back-end sends the result back to the front-end.
(8) The front-end connects GoogleMaps API to display embedded map.
(9) Google Maps API responds with the data.
(10) Front-end displays the traveling plan to user.

Implementation Detail

1. San Francisco, CA: city chosen for this project
   In order to maintain a reasonable size of data without losing generality, we choose San Francisco for our implementation. The reason is mainly because there are a lot of attractions in this city which are suitable for walking. Also, the parking lots are dense as well. Therefore, SF is challenging and interesting for route planning.

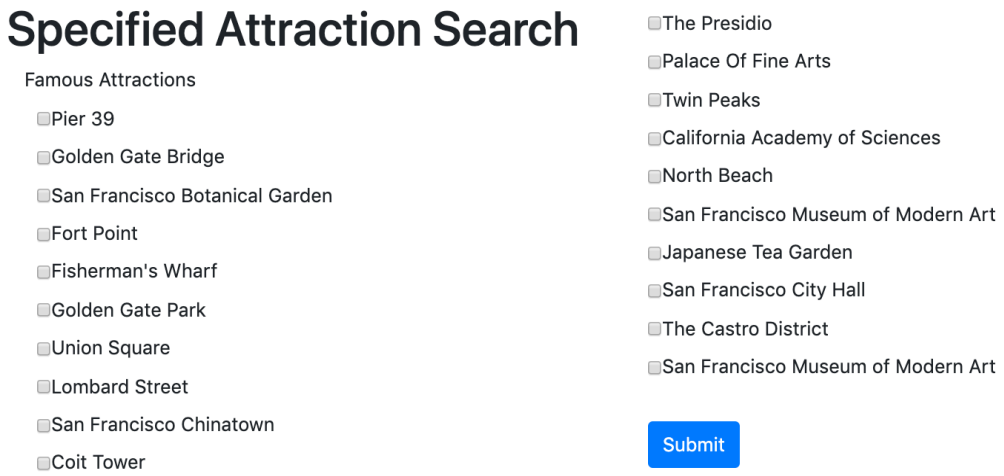
2. Attraction Group: the innovative idea we proposed
   (1) The primary goal for us is to find a best travel plan with parking information. Some attractions are so close to each other that a traveler can park in the same place and walk between the attractions.
   (2) We introduced the concept of “Attraction Group” that is centered with a parking lot. The group includes the attractions of walking distance from the parking lot.
   (3) With the notion of Attraction Group, we can find a parking-oriented travel plan that connects several attraction groups for the drivers.

3. Front-end Interface and Features
   (1) Web Form for Auto Attraction Search



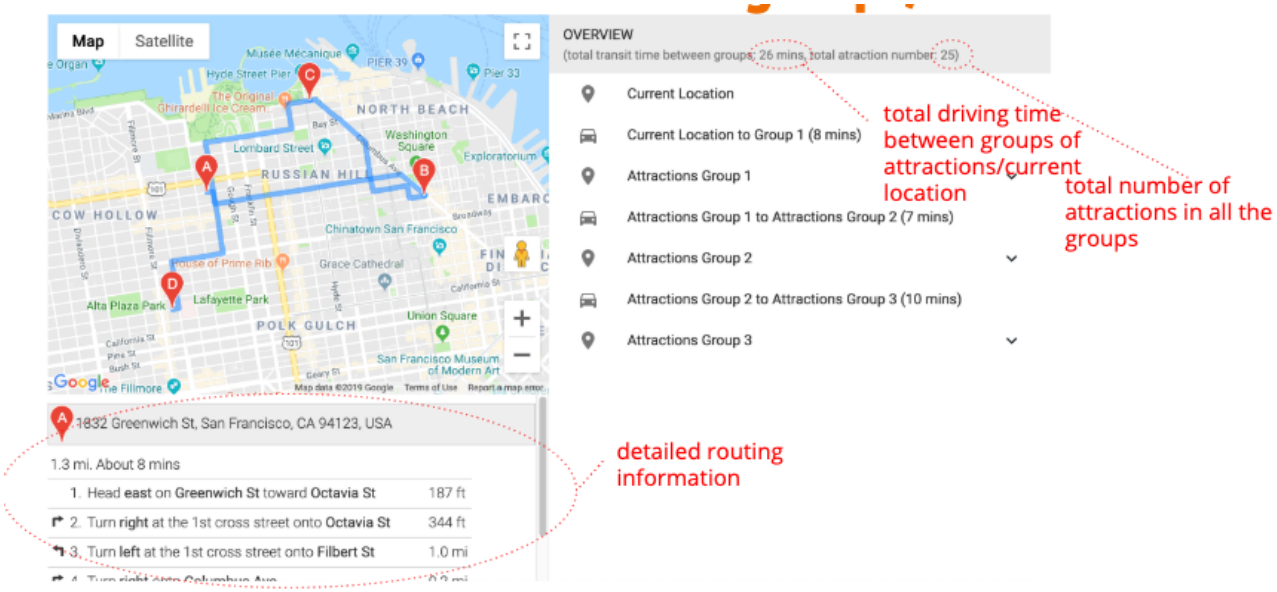
   Upon page loading, front-end fetches the “attraction type” list from back-end to populate the dropdown list. User enters his/her preferred attraction type and the number of parking (k) he/she preferred. The back-end will later return k groups of attractions according to user’s preference, where all attractions in a group share a single parking lot.

(2) Web Form for User-Specified Attraction Search



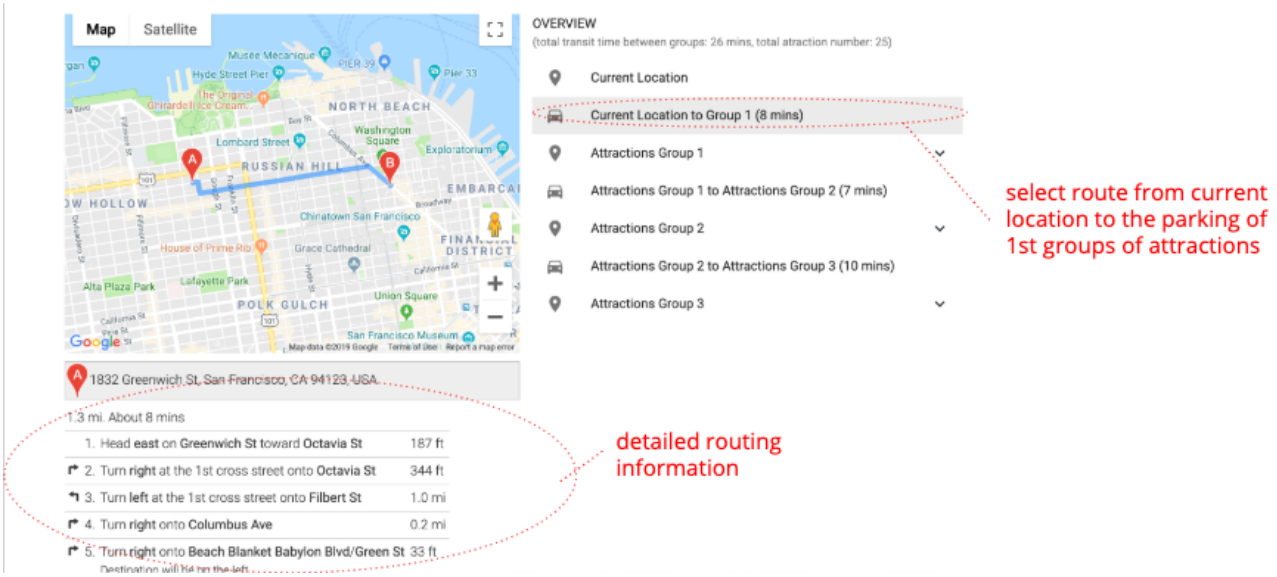
Upon page loading, front-end fetches the “top attractions” list from back-end to populate the checkboxes. User checks the specific attractions he/she preferred. The back-end will later return routing information between attractions and their parking lots.

(3) Overview of routes between groups/current location



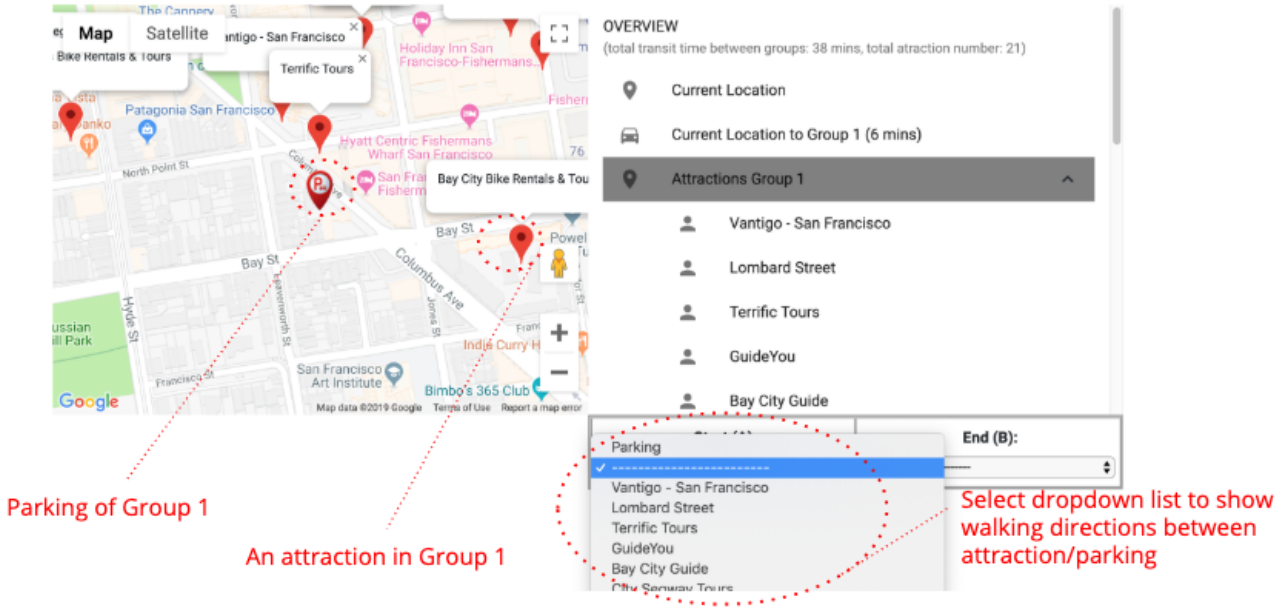
The feature shows user detailed routing information between parking lots of each groups of attractions and current location. It also shows user the calculated total driving time between groups of attractions/current location and shows user the total number of attractions for all the groups.

(4) Driving route between groups/current location



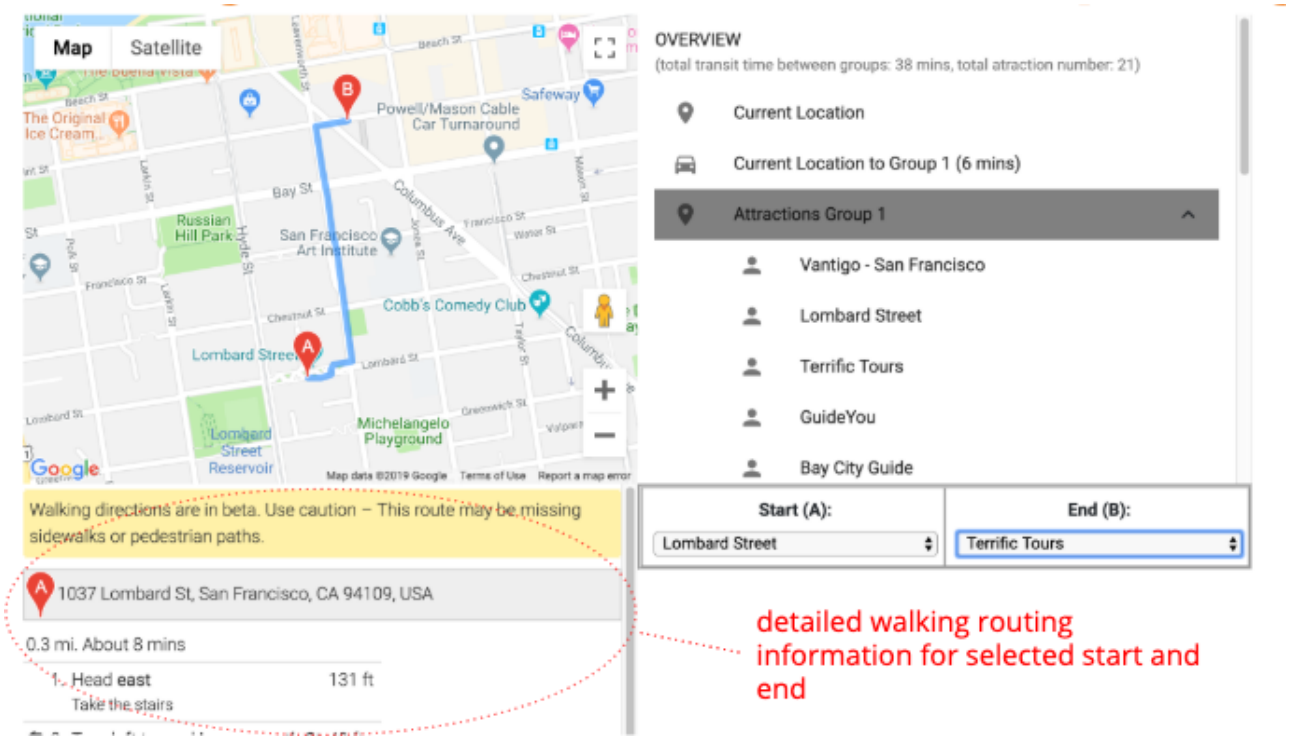
The feature shows user detailed routing information between parking lots of each groups of attractions, or between parking lots of groups of attractions with current location.

(5) Attractions & parking for each group



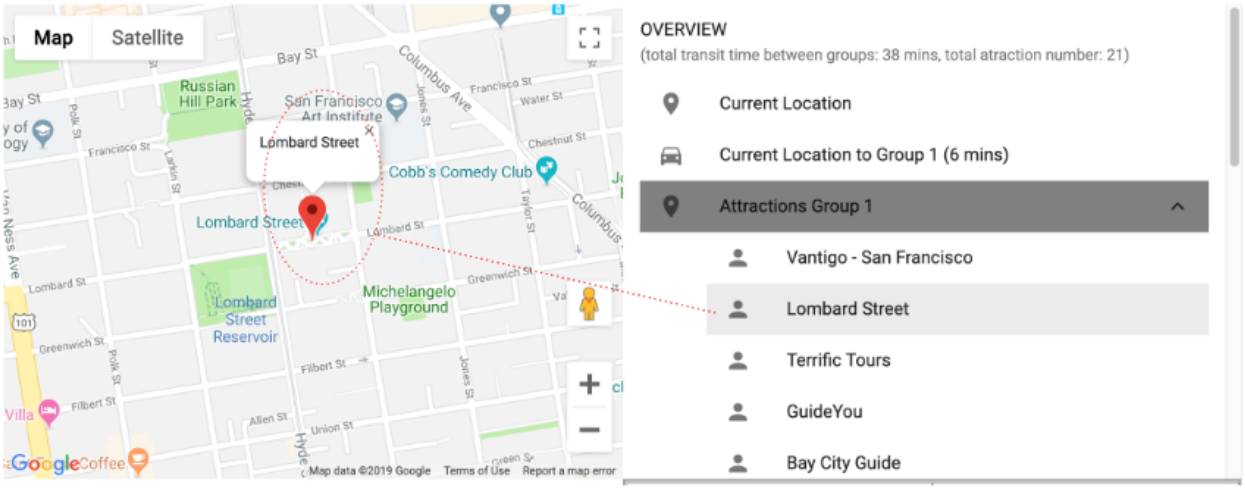
The feature displays markers on map, showing the parking of the group and the attractions in the group.

(6) Walking route between attractions/parking



The feature shows the detailed walking routing information for selected start and end, where the start and end are either parking lot of the group, or an attraction in the group.

(7) Single attraction view



The feature shows a single attraction on map.

4. Back-end Algorithms and Logics

(1) Auto Attraction Search

(a) Based on user's current location (A) (fetched from Google Geolocation API) and the selected attraction categories, we can get a set of recommended attractions from Yelp Fusion API.

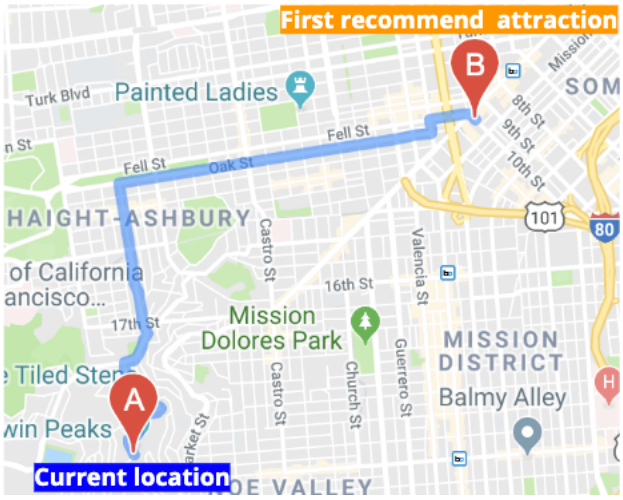
(b) Of the attractions that match the user's preference, we choose the nearest one as our first recommended attraction (B) (graph 1).

(c) Find the nearest parking lot using Google Place API (graph 2).

(d) Centered at the parking lot, search other attractions within walking distance (using Yelp API again). Include all the attractions and the centered parking lot to be an attraction group (graph 3).

(e) Set user's current location at the parking lot of the attraction group 1, repeat (a) to (d).

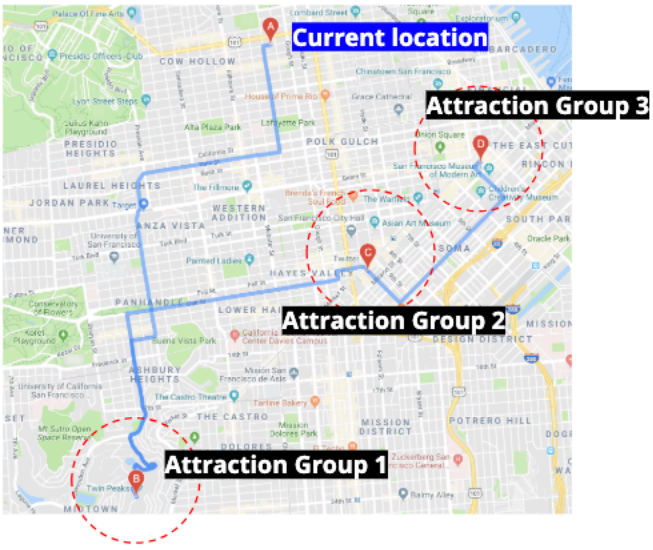
(f) Return a route that connects all attraction groups and use Google distance matrix API to calculate the travel time between each attraction group (graph 4).



Graph 1: Find the nearest attraction

Graph 2: Find a parking lot

Graph 3: Build an attraction group



Graph 4: Calculate the travel route

(2) Specified Attraction Search

(a) Based on user's current location (fetched from Google Geolocation API) and the indicated attractions, we can find a nearest parking lot for each attraction using Google Place API (graph 5).

(b) Traverse through the parking lots found for each attraction. If there are multiple attraction with the same parking lot, we can combine them into a single attraction group (graph 6).

(c) Choose the nearest attraction group as the first one to go iteratively to determine the order to visit (graph 7).

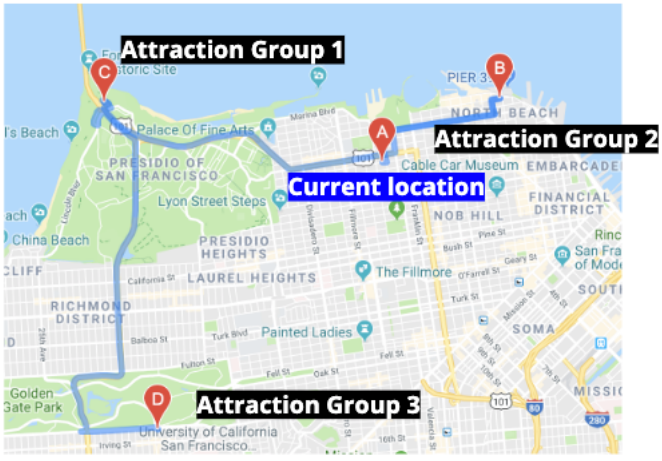
(d) Same as the auto attraction search, return a route that connects all attraction groups and use Google distance matrix API to calculate the travel time between each attraction group (graph 8).

Graph 5: Find the nearest parking lot

Graph 6: Group attractions with same parking lot

```
travel to:
Pier 39
1884.7811774180132(m)
travel to:
Golden Gate Bridge
5575.295440111965(m)
travel to:
San Francisco Botanical Garden
4686.158166719054(m)
```

Graph 7: Iteratively determine the travel order



Graph 8: Calculate the travel route

5. Database
   (1) Our database is used for getting choices and list items for the front-end web form, which stores the attraction categories and our-manually selected famous attractions.
   (2) When the front-end initializes, it sends a GET request to server to get the data to construct the form items/choices. This dynamic setting is really convenient for us to update form choices without re-deploying the application.

	_id ObjectId	place_id String	name String	alias String
1	5cba0736bc085227e4a4b8b1	"B5e069dbuKZG9IJypIhkyg"	"Pier 39"	"pier-39-san-francisco"
2	5cba0771bc085227e4a4b8b2	"Lz7ydUVmsr2DekT9enEhvw"	"Golden Gate Bridge"	"golden-gate-bridge-san-franc
3	5cba0774bc085227e4a4b8b3	"wHhA0cdo0Qzugga1LupN8A"	"San Francisco Botanical Garc	"san-francisco-botanical-garc
4	5cba0775bc085227e4a4b8b4	"kiUnW44GC-nNaRXpAeZ6NQ"	"Fort Point"	"fort-point-san-francisco"
5	5cba0777bc085227e4a4b8b5	"6ymftDVi1J7ilrx_BuY8DA"	"Fisherman's Wharf"	"fishermans-wharf-san-francis
6	5cba0bc0bc085227e4a4b8b6	"KIVq5BlJKTipwtB3NATX0g"	"Golden Gate Park"	"golden-gate-park-san-francis
7	5cba0bc2bc085227e4a4b8b7	"zB4lcBRgKSaPNkdc_D0pKQ"	"Union Square"	"union-square-san-francisco-
8	5cba0bc4bc085227e4a4b8b8	"zTIN_rpW3koEcoEmGN3DwA"	"Lombard Street"	"lombard-street-san-francisc
9	5cba0bc6bc085227e4a4b8b9	"MY7BRqN17bvzD-O_-Mkpuw"	"San Francisco Chinatown"	"san-francisco-chinatown-san
10	5cba0bc7bc085227e4a4b8ba	"efJYZuvYICyzJJYBO-sGvw"	"Coit Tower"	"coit-tower-san-francisco"

Top tourist Attractions

	_id ObjectId	name String	alias String
1	5cb9ff3ebc085227e4a4b89f	"Active Life"	"active"
2	5cba012cbc085227e4a4b8a4	"Arts & Entertainment"	"arts"
3	5cba016fbc085227e4a4b8a5	"Beauty & Spas"	"beautysvc"
4	5cba0175bc085227e4a4b8a6	"Bicycles"	"bicycles"
5	5cba0177bc085227e4a4b8a7	"Education"	"education"
6	5cba017abc085227e4a4b8a9	"Mass Media"	"massmedia"
7	5cba0215bc085227e4a4b8aa	"Nightlife"	"nightlife"
8	5cba028ebc085227e4a4b8ab	"Event Planning & Services"	"eventservices"
9	5cba02bbbc085227e4a4b8ac	"Food"	"food"
10	5cba02e5bc085227e4a4b8ad	"Religious Organizations"	"religiousorgs"
11	5cba0394bc085227e4a4b8ae	"Restaurants"	"restaurants"
12	5cba03cabc085227e4a4b8af	"Shopping"	"shopping"

Attraction Categories

6. The list of APIs the backend used:
   (1) Yelp Fusion API (https://api.yelp.com/v3/businesses): used for attraction search.
   (2) Google Place API (https://maps.googleapis.com/maps/api/place/nearbysearch/): used for parking lot search.
   (3) Google Distance Matrix API (https://maps.googleapis.com/maps/api/distancematrix/): used for traveling time estimation.

Work Distribution

1. Ting-Yu Kang:
   Back-end server, Database, attractions selection, parking lot finding, attraction groups and traveling rout algorithms, API data integration.
2. Yingqiong Shi:
   Front-end interface, Google Map API and embedded map, visual components design.

Challenges Faced and Overcome

1. Scale of the project was too large:
   Our initial goal was to find a trip planner for all kind of transportation method, which includes driving, walking and public transportation. However, after discussing with our professor, we realized that within a semester, it is hard to accomplish a polished project for all the travel modes. It would be better if we could only focus on a single aspect. Therefore, we decided to focus our approach on planning the trip for the user who drives. While driving, parking would be an unignorable task. Accordingly, we put many efforts on integrating parking feature within our app.

2. Attractions are too dense:
   Our initial approach for the problem is to find a parking lot for each attraction, and then provide driving directions between parking lots, walking directions between parking lot and attractions.

This approach seems well but has a significant flaw in San Francisco. In fact, the attractions in SF are denser than parking lots, causing weird routes after showing the results on the map. After realizing this approach did not work well, we start to develop another route-finding algorithm from scratch.

The reason we chose San Francisco as our targeted city is because we believe such a big city would have full of attractions. However, we didn't expect such a high density of attractions. In San Francisco, famous attractions are very close to each other. It is a high probability that several attractions share a single parking lot. To illustrate, if attraction A and B are close enough to share a single parking lot, it does not make sense for user to always park the car first before going to the next attraction. Instead, the user should just walk from A to B.

The solution we found is to put close attractions sharing a single parking lot into a group. For attractions within the group, we provide user details to walk between attractions or from an attraction to parking lot. Between attraction groups, we showed users driving direction between parking lots of two groups. The result makes more sense because after visiting all the attractions in a single group, the user could get on his/her car from the parking lot to go to the next group of attractions.

3. Yelp API does not return famous attractions:
   To accomplish the second mode: "User-Specified Attraction Search", we need to first provide users a list of top attractions in San Francisco so that users could select their preferred attractions from the list. Our initial approach for retrieving the list of top attractions was to fetch the top attractions from Yelp Fusion API. However, after viewing the attractions recommended from Yelp API, many of them are not the most famous attractions in San Francisco. Since our goal is to find a limited number of top attractions for user, we decided to manual select top attractions that everyone is familiar with.

   We searched online to create a list of 20 most famous attractions in San Francisco. After that, we got the location information and descriptions through Yelp API to make our top attraction list.

   Our current approach is a viable solution for our current project. However, after discussing with Professor Pu, we realized that there are some better approaches. We could consider integrate Google API on top of Yelp API to retrieve the name of top attractions. Another approach is to replace the checkboxes with an auto-complete search field for users to type their favorite places. Google API has such a feature and we can utilize it.

Further Ideas

1. For the future improvement, we want to include more cities for our app. Each city has different densities of attractions and parking situations. To include more cities in the future, we need to modify or customize the route-finding algorithm for each city.

2. Our current approach only shows parking lot locations by Google API. With integrating more parking APIs such as ParkWhiz, we could include more useful information in our application, such as on-street parking, parking rate or real-time availability.

3. In the future, we could also integrate more travel modes into our application, such as walking or public transportation. We could also a hybrid transportation in our app. For example, in hybrid transportation mode, user could drive their car to a subway station and then take the subway.

4. For further improvement, we could include food options for users in our application. Users could select the type of food they like and the price range for food and we could integrate the restaurants we recommended for them in our trip.

Related Work

1. Inspirock(https://trip-planner.visittheusa.com/) is an online trip planner that can help the user build a personalized sightseeing itinerary. To plan a trip with Inspirock, the user would input the city he or she is going, dates of the trip, and sightseeing preferences. The planner would then return the user a timeline, a schedule, and a map for a detailed sightseeing itinerary. It also allows the user to edit the returned trip.
   However, Inspirock does not provide user option to park their car.

2. Roadtripper (https://roadtrippers.com/) is a website (mobile app as well) that helps users plan road trips. To schedule a trip with Roadtripper, the user would input the starting location and destination of the journey. The application then generates the route of the road trip. On the map, the app shows the tourist attractions along the way. The user could add any sites into their journey. Roadtripper allows the user to customize their meal options. However, it only lets users discover interesting places during their trip. The application does not generate a recommended itinerary for users.

3. ParkWhiz (parkwhiz.com) is an app for user to find parking at major cities. They provide nearby parking information for user specified location. The parking information includes the distance to the current location, the price of the parking, rating, and etc. The user could also book the parking on their app. ParkWhiz has detailed and complete parking information. They also provide free API for development. In the future, it would be a good idea for our app to integrate the API to get better parking information.