

Homework 6: Travel and Entertainment Search Server-side Scripting

1. Objectives

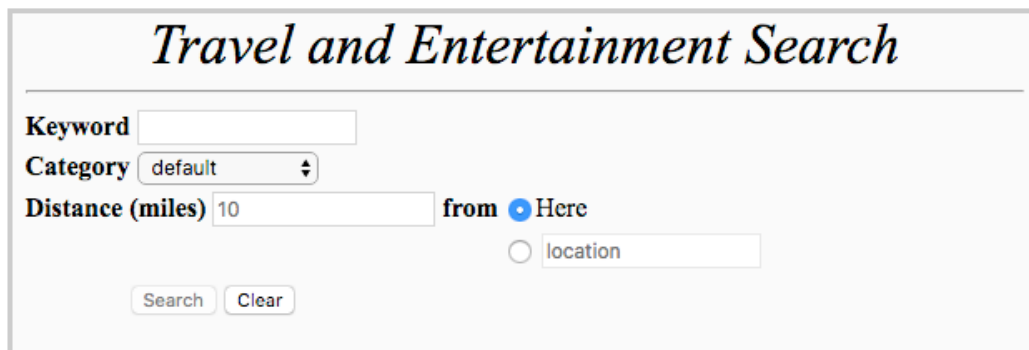
- Get experience with the PHP programming language;
- Get experience with the Google Places API;
- Get experience using JSON parsers in PHP and JavaScript.

2. Description

In this exercise, you are asked to create a webpage that allows you to search for places information using the *Google Places API*, and the results will be displayed in a tabular format. The page will also provide reviews and photos for the selected place.

2.1. Description of the Search Form

A user first opens a page, called **place.php** (or any valid web page name). You should use the *ip-api.com HTTP API* (See hint 3.3) to fetch the user's geolocation, after which the search button should be enabled (it is initially greyed out and disabled when the page loads). The user must enter a keyword and choose what *Category* of place he/she wants to search (categories include [a](#) cafe, bakery, restaurant, beauty salon, casino, movie theater, lodging, airport, train station, subway station, bus station) from a drop-down list. The default value for the "Category" drop-down list is "default", which covers all of the "types" provided by the *Google Places API*. Also, the user can choose the distance (in miles), which is the radius for the search where the center is "Here" (the current location returned from *freegeoip.net HTTP API*) or the location listed in the edit box. When the "Here" radio button is selected, the location edit box must be disabled. When the location edit box is selected, it is a required field, and a location string must be entered. The default distance is 10 miles from the chosen location. Use HTML5 "placeholder" to show the string "location" in the location edit box and "10" in the Distance edit box as the initial values. An example is shown in Figure 1.



Travel and Entertainment Search

Keyword

Category

Distance (miles) from ☒ Here ☐ location

Figure 1(a): Initial Search Screen (Search button disabled)

Figure 1(b): Search Screen (after fetched location)

The search form has two buttons:

- **Search** button: The button must be disabled while the page is fetching the user's geolocation and must be enabled once the geolocation is obtained. An example of valid input is shown in Figure 2. Once the user has provided valid input, your client script should send a request to your web server script **place.php** with the form inputs. You can use either GET or POST to transfer the form data to the web server script. The PHP script will retrieve the form inputs, reformat it to the syntax of the API and send it to the *Google Places API* nearby search service. If the user clicks on the search button without providing a value in the "Keyword" field or "location" edit box, you should show an error "tooltip" that indicates which field is missing. Examples are shown in Figure 3a and 3b.
- **Clear** button: This button must clear the result area (below the search area) and set all fields to the default values in the search area. The clear operation must be done using a JavaScript function.

Figure 2: An Example of valid Search

Travel and Entertainment Search

Keyword

Category

Distance (miles) from ☒ Here ☐ location

Please fill out this field.

Figure 3(a): An Example of Invalid Search (empty input)

Travel and Entertainment Search

Keyword

Category

Distance (miles) from ☐ Here ☒ location

Please fill out this field.

Figure 3(b): An Example of Invalid Search (empty location)

2.2 Displaying Places Results Table

In this section, we outline how to use the form inputs to construct HTTP requests to the Google Places API service and display the result in the web page.

The *Google Places API* is documented here:

<https://developers.google.com/places/>

If the location edit box is selected, the PHP script (i.e., **place.php**) uses the input address to get the geocoding via *Google Maps Geocoding API*. The *Google Maps Geocoding API* is documented here:

<https://developers.google.com/maps/documentation/geocoding/start>

The Google Maps Geocoding API expects two parameters:

- **Address:** The street address that you want to geocode, in the format used by the national postal service of the country concerned. Additional address elements such as business names and unit, suite or floor numbers should be avoided.
- **Key:** Your application's API key. This key identifies your application for purposes of quota management. (Explained in Section 3.1)

An example of an HTTP request to the Google Maps Geocoding API, when the location address is “University of Southern California, CA” is shown below:

```
https://maps.googleapis.com/maps/api/geocode/json?address=University+of+Southern+California+CA&key=YOUR_API_KEY
```

The response includes the latitude and longitude of the address. Figure 4 shows an example of the JSON object returned in the Google Maps Geocoding API web service response.

The latitude and longitude of the address, combined with other input information, are needed when constructing a restful web service URL to retrieve all entities matching the user query, using the *Google Places API* “Nearby Search” service, documented here:

<https://developers.google.com/places/web-service/search>

The *Google Places API* Nearby Search service expects the following parameters:

- **Key:** Your application's API key. This key identifies your application for purposes of quota management.
- **Location:** The geo-location around which to retrieve place information. The geo-location is specified by latitude and longitude values.
- **Radius:** Defines the distance (in meters) within which to return place results. The maximum allowed radius is 50,000 meters. Note that you need to translate miles to meters for a correct value.
- **Type:** Filtering the results to places matching the specified type. Only one type may be specified (if more than one type is provided, all types following the first entry are ignored).
- **Keyword:** A term to be matched against all content that Google has indexed for this place, including but not limited to name, type, and address, as well as customer reviews and other third-party content.

An example of an HTTP request to the *Google Places API* Nearby Search that searches for the nearby caf  s near the University of Southern California within a 10 miles radius is shown below:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=34.0223519,-118.285117&radius=16090&type=cafe&keyword=usc&key=YOUR_API_KEY
```

Figure 5 shows an example of the JSON response returned by the *Google Places API* Nearby Search web service response. See the documentation on “Search Responses” at the bottom of the Place Search section in the *Google Places API* Developer’s Guide at:

<https://developers.google.com/places/web-service/search>

for the details of the output format returned in search responses.

```
▼ results:
  ▼ 0:
    ▶ address_components:  [...]
    formatted_address:    "Los Angeles, CA 90007, USA"
    ▼ geometry:
      ▼ location:
        lat:              34.0223519
        lng:              -118.285117
        location_type:    "GEOMETRIC_CENTER"
      ▼ viewport:
        ▼ northeast:
          lat:             34.0237008802915
          lng:             -118.2837680197085
        ▼ southwest:
          lat:             34.0210029197085
          lng:             -118.2864659802915
        place_id:         "ChIJ7aVxn0THwoARxKIntFtakKo"
      types:
        0:                "establishment"
        1:                "point_of_interest"
        2:                "university"
    status:               "OK"
```

Figure 4: A sample result of Google Maps Geocoding query

```

▼ results:
  ▼ 0:
    ▼ geometry:
      ▼ location:
        lat: 34.0187057
        lng: -118.2843956
      ▼ viewport: {...}
    icon: "https://maps.gstatic.com...e_api/icons/cafe-71.png"
    id: "7163245ddd3b462ced6f37b3f440f1c9e90a2742"
    name: "USC Law School Cafe"
    opening_hours: {...}
    photos: [...]
    place_id: "ChIJddw0WuLHwoARFZlsBjrcY2E"
    price_level: 2
    rating: 5
    reference: "CmRRAAAAK8xA_6qWuStcpF1j...sIFxqsw_wMlMNArzKZLeJyw"
    scope: "GOOGLE"
    types: [...]
    vicinity: "699 Exposition Boulevard, Los Angeles"
  ▼ 1:
    ▼ geometry:
      ▼ location:

```

Figure 5: A sample result of Google Places Nearby Search query

The PHP script (i.e., **place.php**) should pass the returned JSON object to the client side, or parse the returned JSON and extract useful fields and pass these fields to the client side in **JSON format**. You should use JavaScript to parse the JSON object and display the results in a tabular format. A sample output is shown in Figure 6. The displayed table includes three columns: Category icon, Place Name, and Place Address. If the API service returns an empty result set, the page should display “No records have been found” as shown in Figure 7.





















Category	Name	Address
	University of Southern California	Los Angeles
	USC Dornsife College of Letters, Arts and Sciences	3551 Trousdale Pkwy, Los Angeles
	USC Loker Hydrocarbon Research Institute	837 Bloom Walk, Los Angeles
	USC Department of Public Safety	3667 McClintock Ave, Los Angeles
	USC Marshall School of Business	3670 Trousdale Pkwy, Los Angeles
	USC Rossier School of Education	3470 Trousdale Pkwy, Los Angeles
	USC School of Cinematic Arts	900 W 34th St, Los Angeles
	USC Sol Price School of Public Policy	650 Childs Way, Los Angeles
	USC Thornton School of Music	840 W 34th St, Los Angeles
	USC School of Pharmacy	University of Southern California, 1985 Zonal Avenue, Los Angeles
	USC Newman Recital Hall	700 Childs Way, Los Angeles
	USC School of Architecture	Watt Way, Los Angeles
	USC Gloria Kaufman School of Dance (KDC)	849 W 34th St, Los Angeles
	USC Chan Division of Occupational Science and Occupational Therapy	1540 Alcazar St, Los Angeles
	USC Annenberg School for Communication and Journalism	3502 Watt Way #304, Los Angeles
	USC Roski School of Fine Arts	825 Bloom Walk, Los Angeles
	USC Village	3301 S Hoover St, Los Angeles
	USC International Academy	3415 S Figueroa St, Los Angeles
	USC Admission Center	3607 Trousdale Pkwy, Los Angeles
	Keck School of Medicine of USC	1975 Zonal Ave, Los Angeles

Figure 6: An Example of a Valid Search result

Figure 7: An Example of an Empty Search result

When the search result contains at least one record, you need to map the data extracted from the API result to render the HTML result table as described in Table 1.

HTML Table Column	API service response
Category	The value of the “ <i>icon</i> ” attribute that is part of the “ <i>results</i> ” object.
Name	The value of the “ <i>name</i> ” attribute that is part of the “ <i>results</i> ” object.
Address	The value of the “ <i>vicinity</i> ” attribute that is part of the “ <i>results</i> ” object.

Table 1: Mapping the result from Graph API into HTML table

2.3 Displaying Place Details (Reviews and Photos)

In the search result table, if the user clicks on the name of a place, the page should request the place details using the Google Places API “Place Details”, documented at:

<https://developers.google.com/places/web-service/details>

to retrieve reviews and photo references (used to retrieve photos) about the selected place. As required by Google, this must be done using a server PHP script (i.e., **Place.php**). The request needs two parameters (output should be JSON):

- **place_id:** an id returned as result of the Google Places API nearby search service
- **Key:** Your application's API key. This key identifies your application for purposes of quota management.

An example of an HTTP request to the *Google Places API* Place Details is shown below:

```
https://maps.googleapis.com/maps/api/place/details/json?placeid=ChIJ7aVxnOTHwoARxKIntFtakKo&key= YOUR_API_KEY
```

Figure 8(a) shows a sample response.

```

html_attributions:      []
▼ result:
  ▶ address_components:  [...]
  ▶ adr_address:         "<span class=\"locality\"...untry-name\">USA</span>"
  formatted_address:    "Los Angeles, CA 90007, USA"
  formatted_phone_number: "(213) 740-2311"
  ▶ geometry:           {...}
  ▶ icon:                "https://maps.gstatic.com...api/icons/school-71.png"
  id:                   "b85217d74722f6fec94a4135f209e13092d81a5e"
  international_phone_number: "+1 213-740-2311"
  name:                 "University of Southern California"
  ▼ photos:
    ▼ 0:
      height:            552
      ▼ html_attributions:
        ▼ 0:             "<a href=\"https://maps.google.com/maps/contrib/11264996441...33450\"
      ▼ photo_reference:  "CmRaAAAAAYgURxKzKYpyZVC9YBfm1qhonccR-TCH0-f-k5K6FqYwm1RgyN7...wTrOw
      width:             750
    ▶ 1:                 {...}
    ▶ 2:                 {...}
    ▶ 3:                 {...}
    ▶ 4:                 {...}
    ▶ 5:                 {...}
    ▶ 6:                 {...}
    ▶ 7:                 {...}
    ▶ 8:                 {...}
    ▶ 9:                 {...}
  place_id:             "ChIJ7aVxnOTHwoARxKIntFtakKo"
  price_level:          2
  rating:               4.5
  ▶ reference:          "CmRSAAAAuOTb_1Am1Vbr9lva...ZRPXBWehd_Y03eSbGwu2QwA"
  ▼ reviews:
    ▼ 0:
      author_name:       "Ambar Pal"
      ▶ author_url:      "https://www.google.com/m...244132951202177/reviews"
      language:          "en"
      ▼ profile_photo_url: "https://lh3.googleusercontent.com/-xNk-Ah0clEs/AAAAAAAAAI/AAAA
      rating:            5
      relative_time_description: "2 months ago"
      ▼ text:            "I had a great great time here. The campus is quite intricately m
        running tracks, swimming pools. The Doheny Library is one of a ki
      time:              1512277072

```

Figure 8(a): An example of a Place Details search result (Keyword: USC, Category: default)

Once the PHP script retrieves the response from the *Google Places API* Place Details query, you should use a PHP script to parse the JSON object and use photos' reference (photo_reference field) to request high-resolution photos via *Google Places API* "Place Photos" query, as documented here:

<https://developers.google.com/places/web-service/photos>

The request requires providing values for three parameters:

- **Photo_reference:** A string identifier that uniquely identifies a photo. Photo references are returned from either a Place Search or Place Details request.
- **Key:** Your application's API key. This key identifies your application for purposes of quota management.
- **maxheight or maxwidth:** specifies the maximum desired height or width, in pixels, of the image returned by the Place Photos service.

An example of an HTTP request to the *Google Places API* Place Photos is shown below:

```
https://maps.googleapis.com/maps/api/place/photo?maxwidth=750&photoreference=CnRtAAAATLZNl354RwP_9UKbQ_5Psy40texXePv4oAlgP4qNEkdIrkyse7rPXYGd9D_Uj1rVsQdWT4oRz4QrYAJNpFX7rzqqMlZw2h2E2y5IKMUZ7ouD_SlcHxYq1yL4KbKUv3qtWgTK0A6QbGh87GB3sscrHRIQiG2RrmU_jF4tENr9wGS_YxoUSSDrYjWmrNfeEHSGSc3FyhNLlBU&key=YOUR_API_KEY
```

The response is a binary image file. You should save the top 5 high-resolution photos on the server side using the function “`file_put_contents()`”. Figure 8(b) shows a sample response file.

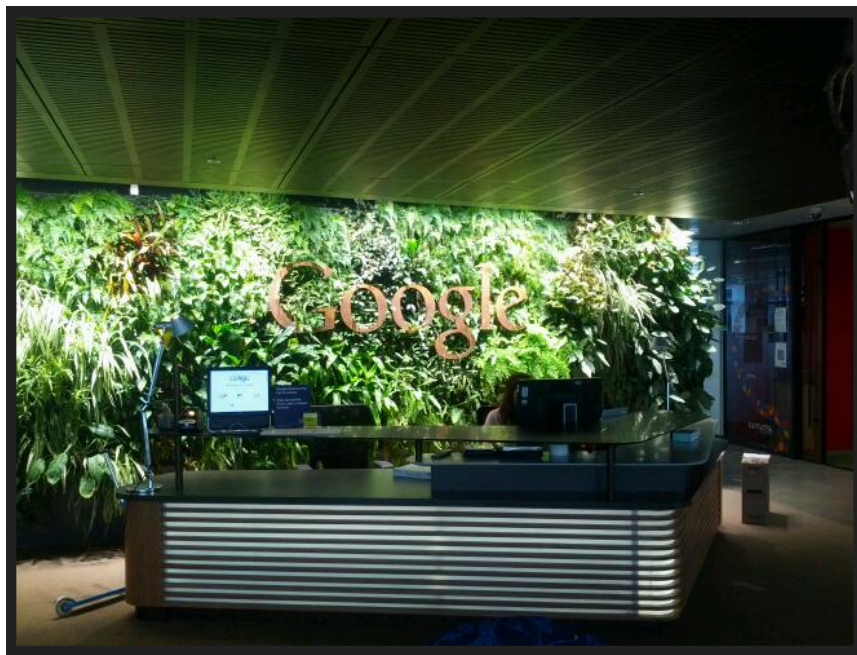


Figure 8(b): An example of Place Photos query response

After retrieving place details data, you can pass the place detail JSON object to the client side and show the top 5 reviews and top 5 photos in a tabular format.

At the top of reviews, you also need to show the name of the selected place.

Note you must only show the photos saved on your server machine. Calling directly to Google Place Photo API at client side is not allowed.

Table 2: Mapping the result from Google Places API into HTML Table

HTML Table Column	API service response
Review	You should display at most 5 reviews present in data attribute, which is present in “ <i>reviews->text</i> ” attribute. Each review should contain the author name and author’s profile photo.
Photo	You should display at most 5 photos using the photo_reference, which is present in “ <i>photos</i> ” attribute.

The details information includes two sub-sections: Reviews and Photos which are by default hidden (i.e., collapsed) (as shown in Figure 9).

The details information should over-write the result table and needs to be displayed under the

search form. When the user clicks on the  button, the “Reviews” sub-section should be expanded, and the “Photos” sub-section should be hidden (if it is open) and vice versa.

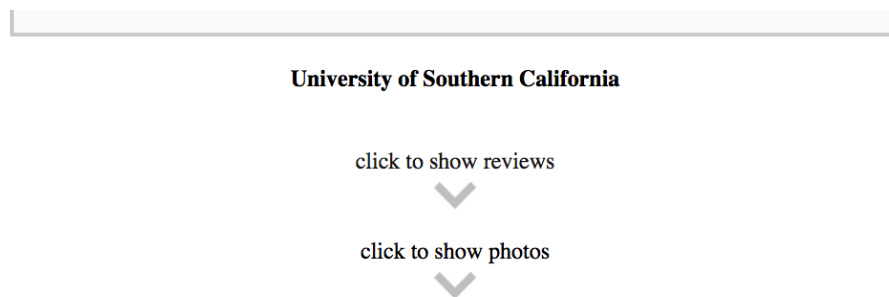


Figure 9: Both the reviews and photos are hidden

The “Reviews” sub-section should display the top 5 reviews, as shown in Figure 10. Each review should display the author’s name and his/her profile picture and his/her reviews text.

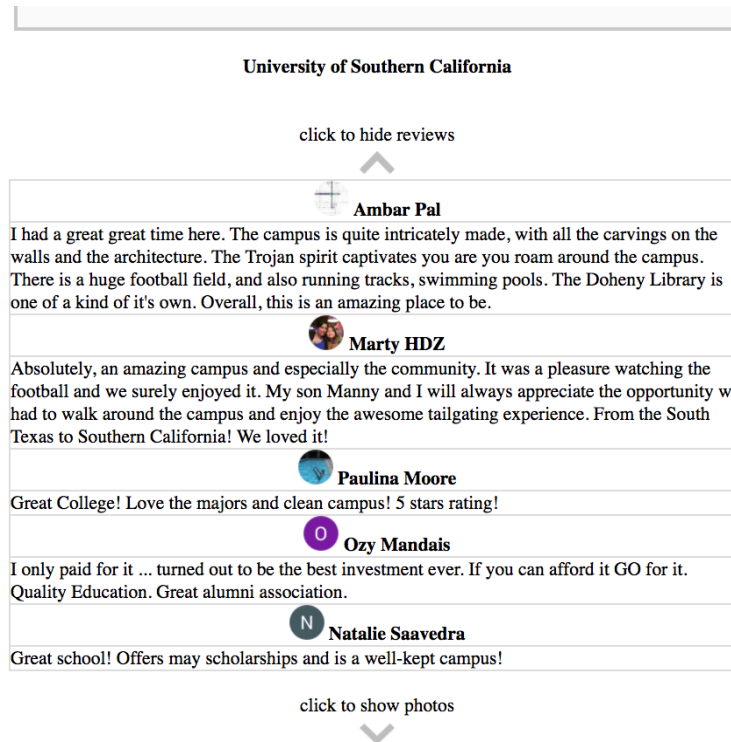


Figure 10: When reviews are clicked, photos are hidden.

The “Photos” sub-section should display a maximum of 5 photos (as shown in Figure 11). If any photo is clicked, it should be opened in a new tab with the original high-resolution photo saved on the server side.

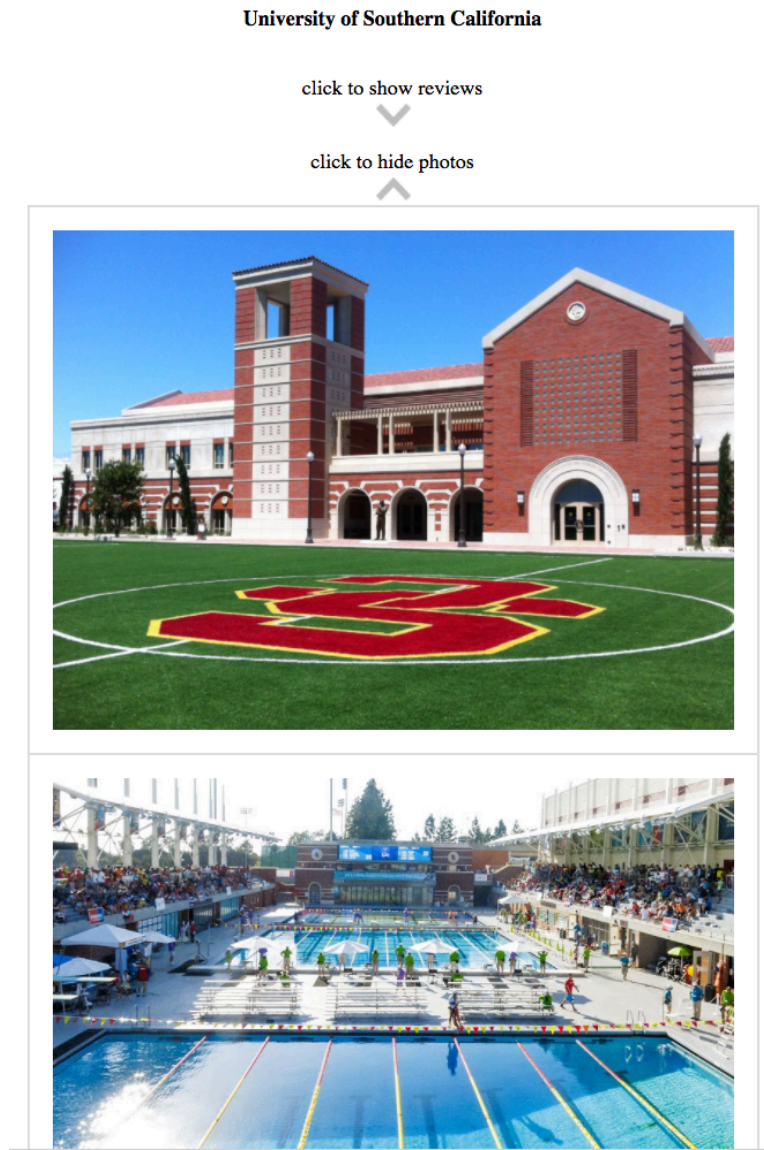


Figure 11: When photos are clicked, reviews are hidden.

If the API service returns an empty result set, the page should display “No Reviews Found” instead of review section and “No Photos Found” instead of photo section. A sample output is shown in Figure 12 and Figure 13.



Figure 12: When no reviews are found.

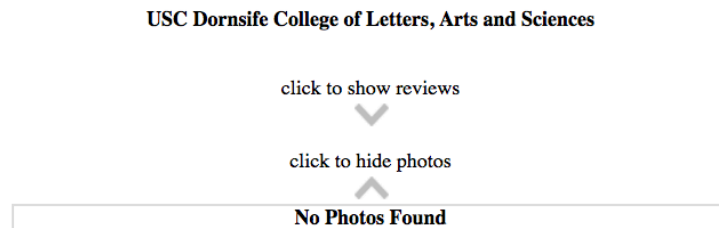


Figure 13: When no photos are found.

Note that:

- You must use PHP to request all JSON objects except when calling the *ip-api.com API* which should be called on the client side using JavaScript.
- Expanding or hiding sub-areas should be implemented using JavaScript and you are not allowed to use JQuery.
- High-resolution images must be called using PHP and photos must be saved on the server-side machine. The client (JavaScript) will access the photos saved on the server.

2.4 Displaying Map and Directions

In the search result table, when the corresponding address of a certain record is clicked, a Google Map with a marker of the place should pop up. If the Google Map is already displayed, clicking it will make the map hidden again. The map should not over-write the result table and needs to be displayed right under the address that you click on. Please see the video for the details.

You should use the Google Maps JavaScript Library, documented at:

<https://developers.google.com/maps/documentation/javascript/adding-a-google-map> to construct the map.

A sample is shown in Figure 14 when selecting the address “699 Exposition Blvd, Los Angeles” of USC Law School Cafe.







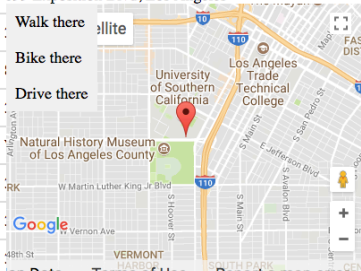





Category	Name	Address
	USC Law School Cafe	699 Exposition Blvd, Los Angeles
	Cafe Dulce (USC Village)	Walk there 
	Good Karma Cafe	Bike there 
	Tutor Hall Café	Drive there 
	Little Galen	
	Fertitta Cafe	
	Urbrnmrkt	
	Literatea	

Figure 14: Maps shown when clicking the address of a record.

At the top left corner of the map, there should be a travel mode list (including Walk there, Bike there, and Drive there). If a user clicks on an option, the Google Map with a Marker should be replaced by a Google Map with directions from the location that you choose as the “center point” on the search form to the selected record on a Google Map. A sample is shown in Figure 15 when choosing “Walk there” based on Figure 14.

You need to use Direction service to construct the direction route map (<https://developers.google.com/maps/documentation/javascript/directions>).

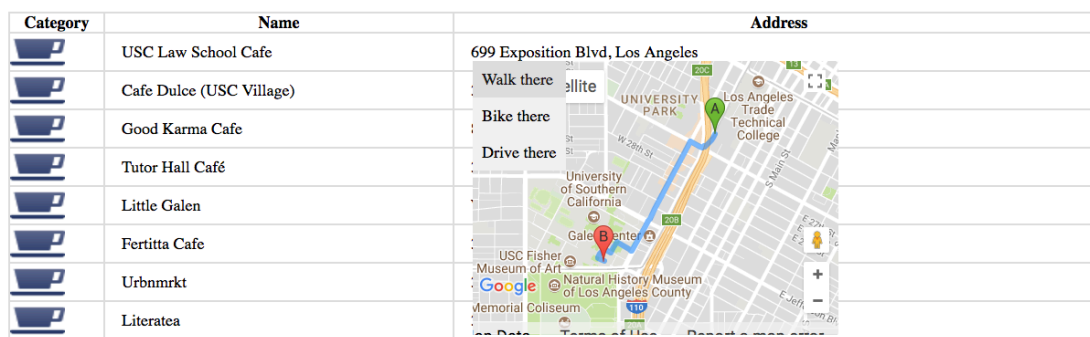


Figure 15: the directions after clicking “Walk there”

2.5 Saving Previous Inputs

In addition to displaying the results, the PHP page should maintain the provided values. For example, if a user searches for “Keyword: USC, Category: café, Distance: 15 from Here”, the user should see what was provided in the search form when displaying the results. In addition, when clicking on a “Name”, the page should display the reviews/photos and keep the values provided in the search form. It follows that you need to keep the whole search box/input fields and buttons even while displaying results/errors.

In summary, the search mechanism to be implemented behaves as follows:

- Based on the query in the search form, construct a web service URL to retrieve the output from the *Google Places API* service.
- Pass the (possibly edited) JSON to the client side and parse JSON using JavaScript.
- Display the places information in tabular and chart formats.
- Display the reviews and photos in tabular formats.
- Display the map and directions.

3. Hints

3.1 How to get Google API Key

- To get a Google API key, please follow these steps:
- Go to the Google Developers Console:

https://console.developers.google.com/flows/enableapi?apiid=geocoding_backend&keyType=SERVER_SIDE&reusekey=true

- Create a project.
- At every Google APIs' guide page, click "Get a key" and select a created project.

Note that you should NOT use a google account associated with a USC e-mail.

3.2 Google Maps JavaScript API on demand API Documentation

- Adding a Google Map with a Marker to Your Website:

<https://developers.google.com/maps/documentation/javascript/adding-a-google-map>

- Directions Service:

<https://developers.google.com/maps/documentation/javascript/directions>

3.3 Get geolocation using IP-API.com

You need to use *ip-api.com* for searching the geolocation based on IP addresses. An example call looks like:

<http://ip-api.com/json>

The response is a JSON object shown in Figure 16.

```
as: "AS20001 Time Warner Cable Internet LLC"
city: "Los Angeles"
country: "United States"
countryCode: "US"
isp: "Time Warner Cable"
lat: 34.0266
lon: -118.2831
org: "Time Warner Cable"
query: "104.32.172.65"
region: "CA"
regionName: "California"
status: "success"
timezone: "America/Los_Angeles"
zip: "90007"
```

Figure 16: Response from *ip-api.com* API

This article introduces some similar APIs, so you have more choice for your hw6 :

<https://ahmadawais.com/best-api-geolocating-an-ip-address/>

Use of Freegeoip API is not recommended.

3.4 Parsing JSON-formatted data in PHP

In PHP 5, you can parse JSON-formatted data using the “*json_decode*” function. For more information, please go to <http://php.net/manual/en/function.json-decode.php>.

You can encode data into JSON-formatted objects using the “*json_encode*” function. For more information, please go to <http://php.net/manual/en/function.json-encode.php>.

3.5 Read and save contents in PHP

To read the contents of a JSON-formatted object, you can use the “*file_get_contents*” function. To save contents on the server side, you can use “*file_put_contents*” function.

4. Files to Submit

In your course homework page, you should update the **Homework 6 link** to refer to your new initial web page for this exercise. Also, submit your source code file (**it must be a single .php file**) electronically to the csci571 account so that it can be graded and compared to all other students’ code via the MOSS code comparison tool.

****IMPORTANT**:**

- All discussions and explanations in Piazza related to this homework are part of the homework description and grading guidelines. So please review all Piazza threads, before finishing the assignment. If there is a conflict between Piazza and this description and/or the grading guidelines, **Piazza always rules.**
- You should not use JQuery for Homework 6.
- You **should not call the Google Places APIs (including nearby search service, place detail service, and place photo service) directly from JavaScript**, bypassing the Apache/HTTP proxy. Implementing any one of them in JavaScript instead of PHP will result in a **4-point penalty.**
- The link to the video is: <https://youtu.be/1uffTsR2jLk>