

# Homework 6: Server-side Scripting using PHP, JSON, and Facebook Graph API

## 1. Objectives

- Get experience with the PHP programming language.
- Get experience with the Facebook Graph API.
- Get experience with the Google Geocoding API.
- Get experience with the JSON parsers in PHP.

## 2. Description

In this exercise, you are asked to create a webpage that allows you to search for information about a set of Facebook entities (i.e., users, pages, events, places, and groups) using the Facebook Graph API, and the results will be displayed in a tabular format.

Please note that to use the Facebook Graph API, you need to have an App Secret, App ID and Access token as explained in detail in Sections 3.1 and 3.2.

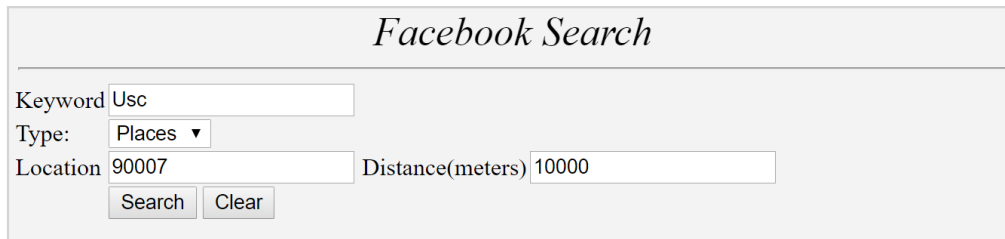
### 2.1. Description of the Search Form

A user first opens a web page, called **Search.php (or any valid web page name)**, where he/she can enter a keyword and choose an entity type (i.e., users, pages, events, places and groups) from a drop-down list. The default value for the “Type” drop-down list is *Users*. An example of the web page is shown in Fig. 1. If the user chooses the “*Places*” option from the “Type” drop-down list, the page shows an additional two text fields (Location and Distance). An example of the web page when searching for Places is shown in Fig. 2. Showing/hiding the location and distance fields should be implemented in JavaScript based on the Type selection.



The image shows a web form titled "Facebook Search". It contains a "Keyword" text input field with the value "Usc". Below it is a "Type:" label followed by a dropdown menu currently showing "Users". At the bottom of the form are two buttons: "Search" and "Clear".

**Figure 1:** Initial Search Screen (keyword: USC, Type: Users)



The image shows the same "Facebook Search" web form as in Figure 1, but with different values. The "Keyword" field still has "Usc". The "Type:" dropdown menu now shows "Places". Below the dropdown, there are two new text input fields: "Location" with the value "90007" and "Distance(meters)" with the value "10000". The "Search" and "Clear" buttons remain at the bottom.

**Figure 2:** Initial Search Screen (keyword: USC, Type: Places, Location 90007, Distance: 1000)

The search form has two buttons:

- The **Search** button: If the user clicks on the Search button without providing a value for all the form fields, you should show an HTML5 error message. The error message **should indicate which fields** are missing. An example of an error message is shown in Fig. 3.

The image shows a web form titled "Facebook Search". It contains a "Keyword" text input field, a "Type:" label, and two buttons labeled "Search" and "Clear". A red rectangular border highlights the "Keyword" input field. A white tooltip box with a grey border is positioned over the bottom-left corner of the "Keyword" field, containing the text "This cant be left empty".

**Figure 3:** An Error Message when there is no input

Once the user provides all required selections and data and clicks on the Search button, your page should send a request to your Apache web server for **Search.php** with the form data. You can use either GET or POST to transfer the form data to the web server. A PHP script will retrieve the data and use it to query the *Facebook Graph API* Service.

- The **Clear** button: This button must clear the result area and reset the form fields to their initial state (i.e., empty keyword and the type option should be *Users* and the location and distance text fields should be empty and hidden). The Clear operation should be implemented using a JavaScript function.

## 2.2. Displaying Search Results

In this section, we outline how to use the form data to construct HTTP requests to the *Facebook Graph API* service and display the result in the web page.

The PHP script (i.e., **Search.php**) uses the input information (e.g., Keyword and Type) to construct a restful web service URL to retrieve all entities matching the user query. The *Facebook Graph API* expects four parameters:

- **q**: the search keyword provided by the user in the keyword text field.
- **type**: the entity type which is being searched. This value depends on the Type option selected by user.
  - If the type option is *Users*, the value of the type parameter is *user*.
  - If the type option is *Pages*, the value of the type parameter is *page*.
  - If the type option is *Events*, the value of the type parameter is *event*.
  - If the type option is *Groups*, the value of the type parameter is *group*.
  - If the type option is *Places*, the value of the type parameter is *place*.
- **fields**: The fields to be retrieved for the entities being searched. The fields are comma separated. For example, to retrieve id, name, and picture for the entities which are being

searched, we can set the parameter value as *fields=id,name,picture*. In this homework, we are only interested in the following fields:

- id
- name
- picture
- albums
- posts
- **center:** This parameter is used only when *type=place*. It is used to narrow the search to a specific location. The value is a geo-location specified by latitude and longitude values separated by comma (e.g., *center=34.028418,-118.283953*).
- **distance:** This parameter is used only when *type=place*. It is used to narrow the search to a specific location within radius (i.e., spatial circle query). The value is a number which represents the distance in meters.
- **access\_token:** This is the token which you will get during the registration step (explained in Section 3.2).

An example of the HTTP call to the *Facebook Graph API* when the Type is *user* is shown below.

```
https://graph.facebook.com/v2.8/search?q=The_Keyword_to_be_searched&type=user&fields=id,name,picture.width(700).height(700)&access_token=Your_Access_Token
```

An example of the HTTP call to the Facebook Graph API when the type is *event* is shown below.

```
https://graph.facebook.com/v2.8/search?q=The_Keyword_to_be_searched&type=event&fields=id,name,picture.width(700).height(700)&access_token=Your_Access_Token
```

If the chosen Type is *places*, the PHP script should first translate the location address provided by the user (e.g., *941 Bloom Walk, Los Angeles, CA 90089-0781*) into the corresponding geo-location (i.e., latitude and longitude) before calling the *Facebook Graph API*. The PHP script should construct a web service URL to query the *Google Geocoding API* appropriately:

```
http://maps.googleapis.com/maps/api/geocode/json?address=941+Bloom+Walk,Los+Angeles,+CA+90089-0781&key=YOUR_API_KEY
```

How to create a Google API key is explained in section 3.4. The response of this URL is a JSON-formatted object. Two key pieces of data returned are the latitude and longitude values for the given address. Figure 4 shows an example of the JSON object returned in the *Google Geocoding API* web service response. After extracting latitude and longitude, you can use them to construct another URL to call the *Facebook Graph API* to query *place* objects as shown below.

```
https://graph.facebook.com/v2.8/search?q=The_Keyword_to_be_searched&type=place&center=34.028418,-118.283953&distance=1000&fields=id,name,picture.width(700).height(700)&access_token=Your_Access_Token
```

The API returns the list of objects matching a given input in JSON format. The returned output includes a set of fields for each object (e.g., name, id, and picture). Fig. 5 shows a sample response from API.

```

{
  "results": [
    {
      "address_components": [
        "941 Bloom Walk, Los Angeles, CA 90089, USA",
      ],
      "formatted_address": "941 Bloom Walk, Los Angeles, CA 90089, USA",
      "geometry": {
        "location": {
          "lat": 34.0200998,
          "lng": -118.2892749
        },
        "location_type": "RANGE_INTERPOLATED",
        "viewport": {
          "northeast": {
            "lat": 34.0214418802915,
            "lng": -118.2879302697085
          },
          "southwest": {
            "lat": 34.0187439197085,
            "lng": -118.2906282302915
          }
        }
      },
      "partial_match": true,
      "place_id": "Eto5NDEgQmxvb20gV2FsaywgTG9zIEFuZ2VsZXMsIENBIDkwMDg5LCBVU0E",
      "types": [
        "street_address"
      ]
    }
  ],
  "status": "OK"
}

```

**Figure 4:** A Sample Result of Google Geocoding Query

```

{
  "data": [
    {
      "id": "613411135515447",
      "name": "Usc",
      "picture": {
        "data": {
          "is_silhouette": false,
          "url": "https://scontent.xx.fbcdn.net/v/t1.0-1/p50x50/15284113_586576518198909_3548263184340383065_n.jpg?oh=5a07f1"
        }
      }
    },
    {
      "id": "1913738022192608",
      "name": "Confesiones Usc Palmira",
      "picture": {
        "data": {
          "is_silhouette": false,
          "url": "https://scontent.xx.fbcdn.net/v/t1.0-1/c14.14.177.177/s50x50/1150873_1375271032705979_1306694779_n.jpg?oh=d6375417992f73ad68af8ed8685728e0&oe=5909C29C"
        }
      }
    }
  ],
  "paging": {
    "next": "https://graph.facebook.com/v2.8/search?"
  }
}

```

**Figure 5:** A Sample Response of the search query from the Facebook Graph API


























The PHP script (i.e., **Search.php**) should parse the returned JSON-formatted object and extract the necessary fields. After extracting the data, the PHP script should display the data in a tabular format below the search form. A sample output is shown in Fig. 6. The profile photo and Name should be displayed in the result table. Also for all the types except for events, a “Details” column should be listed in this table. If the API service returns an empty result set, the page should display “*No Records have been found*” instead of the result table, below the Search form. This case is shown in Fig. 7.

*Facebook Search*

---

Keyword

Type:

Profile Photo	Name	Details
	Usc	<a href="#">Details</a>
	Confesiones Usc Palmira	<a href="#">Details</a>
	Uschi Appel	<a href="#">Details</a>
	Uschi Schoenfelder	<a href="#">Details</a>
	Uschi Fellner	<a href="#">Details</a>
	Uschi Graf	<a href="#">Details</a>
	Pamela Uscanga	<a href="#">Details</a>
	Junior Usca Vilca	<a href="#">Details</a>
	Usc Pouyastruc	<a href="#">Details</a>
	Uschi Zeiser-Graf	<a href="#">Details</a>
	Uschi Melly	<a href="#">Details</a>
	Uschi Strahberger	<a href="#">Details</a>
	Uschi Ratey	<a href="#">Details</a>
	Uschi Müller	<a href="#">Details</a>
	Ricardo Uscanga	<a href="#">Details</a>
	Uschi Grote	<a href="#">Details</a>
	Pablo Uscanga Ramon	<a href="#">Details</a>
	Uschi Neske	<a href="#">Details</a>
	Apwu Usc Cba	<a href="#">Details</a>
	Uschi Seiser	<a href="#">Details</a>
	Uschi Hesse	<a href="#">Details</a>
	Susy Uscanga Dominguez	<a href="#">Details</a>
	Audra Uscilaite	<a href="#">Details</a>
	Hilda Uscata Ccaulla	<a href="#">Details</a>
	Usc Courcelloise	<a href="#">Details</a>

**Figure 6:** An Example of Search Result (keyword: *usc*, type: *Users*)



The image shows a Facebook search interface. At the top, it says "Facebook Search". Below that, there is a search bar with the text "Usc ucla ucsd" and a dropdown menu with "Users" selected. There are "Search" and "Clear" buttons below the search bar.

No Records has been found

**Figure 7:** Example of Empty result set

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.

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

HTML Table Column	API service response
Profile Photo	The value of the “ <i>url</i> ” attribute which is part of the “ <i>data</i> ” object which belongs to the “ <i>picture</i> ” object. Each profile photo is about 30x40 in size (px)
Name	The value of the “ <i>name</i> ” attribute
Details	Hyperlink that uses the “ <i>id</i> ” attribute of the user (see section 2.3)

When the profile picture is clicked, the picture is to be shown in a new window with the original size (a high resolution image has to be shown).

**Note: There is no major change when calling the API for Places or Groups. The only change is to provide a different value for the Type parameter while the JSON response is the same for all types**

For the type “*events*”, you need to show the picture, Name of the event and place of the event. The mappings for the events type is as follows

HTML Table Column	API service response
Profile Photo	The value of the “ <i>url</i> ” attribute which is part of the “ <i>data</i> ” object which belongs to the “ <i>picture</i> ” object. Each profile photo is about 30x40 in size (px)
Name	The value of the “ <i>name</i> ” attribute

Place	The value of “ <i>name</i> ” attribute which is present in “ <i>place</i> ” attribute.
-------	--

The call for the type events is as follows.

[https://graph.facebook.com/v2.8/search?q=The\\_keyword\\_to\\_be\\_searched&type=event&fields=id,name,picture.width\(700\).height\(700\),place&access\\_token=Your\\_Access\\_Token](https://graph.facebook.com/v2.8/search?q=The_keyword_to_be_searched&type=event&fields=id,name,picture.width(700).height(700),place&access_token=Your_Access_Token)

A sample response for the above call is as follows:

```
{
  "description": "With Hacktech coming up, we know some of you want to form teams or to brainstorm hack ideas! Invite your friends and start posting on this event!",
  "end_time": "2017-03-05T16:00:00-0800",
  "name": "USC Storms Hacktech",
  "place": {
    "name": "California Institute of Technology - Caltech",
    "location": {
      "city": "Pasadena",
      "country": "United States",
      "latitude": 34.137893863287,
      "longitude": -118.12528024408,
      "state": "CA",
      "street": "1200 E California Blvd",
      "zip": "91125"
    },
    "id": "6391532964"
  },
  "start_time": "2017-03-03T17:00:00-0800",
  "id": "170876643403120"
}
```

**Note:** There is no “Details” field for events.

### 2.3. Displaying Details Information

In the search result table, the Details column contains a “Details” link. When clicking on the “Details” link of a certain record, the PHP script (i.e., **Search.php**) should use the **corresponding** “id” value to make another HTTP request to the restful web service URL to query detailed information about the selected record. In order to retrieve the top 5 albums and top 5 posts, you can use the following format for the query.

[https://graph.facebook.com/v2.8/id?fields=id,name,picture.width\(700\).height\(700\),albums.limit\(5\){name,photos.limit\(2\){name,picture}},posts.limit\(5\)&access\\_token=Your\\_Access\\_Token](https://graph.facebook.com/v2.8/id?fields=id,name,picture.width(700).height(700),albums.limit(5){name,photos.limit(2){name,picture}},posts.limit(5)&access_token=Your_Access_Token)

The above query also requests two photos in each album. In the above API call replace the ‘id’ with the id of the object, as shown in the following example.

https://graph.facebook.com/v2.8/**124984464200434**?  
fields=id,name,picture.width(700).height(700),albums.limit(5){name,photos.limit(2){name,picture}},posts.limit(5)&access\_token=Your\_Access\_Token

Fig. 8 shows a sample response from the ‘id’ API.

```

{
  "id": "124984464200434",
  "name": "USC Trojans",
  "picture": {
    "data": {
      "height": 700,
      "is_silhouette": false,
      "url": "https://scontent.xx.fbcdn.net/v/t1.0-1/14317609_1266608076704728_oh=d191379be5b755b0dbaac41bea70e00c&oe=59458AF8",
      "width": 700
    }
  },
  "albums": {
    "data": [
      {
        "name": "Timeline Photos",
        "photos": {
          "data": [
            {
              "name": "FINAL: USC 92, OSU 66.\n\nTrojans earn their fifth straight longest streak since 2009!\n\n#ItTakesATeam",
              "picture": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16649115_1419767018055499_5986104883150813759_n.jpg?oh=7da6cedbe9cde4a9c0cbece4cafecf7b&oe=593966BB",
              "id": "1419767018055499"
            }
          ],
          "paging": {
            "next": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64",
            "previous": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64"
          }
        }
      }
    ],
    "paging": {
      "next": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64",
      "previous": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64"
    }
  },
  "posts": {
    "data": [
      {
        "message": "FINAL: USC 92, OSU 66.\n\nTrojans earn their fifth straight longest streak since 2009!\n\n#ItTakesATeam",
        "story": "USC Trojans with Smart & Final.",
        "created_time": "2017-02-10T06:10:09+0000",
        "id": "124984464200434_1419767018055499"
      }
    ],
    "paging": {
      "next": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64",
      "previous": "https://scontent.xx.fbcdn.net/v/t1.0-0/s130x130/16387990_1417689174929950_6972417862069664131_n.jpg?oh=2c04c93539a35fadfb8a8ea464873f2f&oe=59486D64"
    }
  }
}

```



**Figure 8:** A Sample Response of the search for pages

The PHP script should map the data retrieved from the API service response to render the object details using the mapping described in Table 2.

**Table 2:** Mapping the result from Graph API into HTML Table

Table Column	API Service Response
Albums	You should display at most 5 albums present in <i>data</i> attribute which is present in <i>albums</i> attribute. Each Album should display two pictures. Each picture must be of size 80x80 (px)
Posts	You should display at most 5 posts present in <i>data</i> attribute which is present in <i>posts</i> attribute.

The details information include two sub-sections: *Albums* and *Posts* which are by default hidden (as shown in Fig. 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 “Albums” hyperlink, the “Albums” sub-section should be expanded and the “Posts” sub-section should be hidden (if it is open) and vice versa.



**Figure 9:** Both the Albums and Posts are hidden

The “Albums” sub-section should display a maximum of 5 albums which are hidden by default (as show in Fig. 11). When clicking on the header of each album, the album displays only two photos (as shown in Fig. 11). When there are no images under an album, the album should not be clickable (it should be just a plain text). If the header of the album is clicked while the album photos are shown the album photos should be hidden. When a photo is clicked, it should be opened in a new window with the original high resolution image. To obtain original high resolution images, the following API can be used.

`https://graph.facebook.com/v2.8/id/picture?access_token=Your_Access_Token`

Here ‘id’ is the picture id obtained in Fig 8. This ‘id’ will be seen in Albums->data->photos->data. It will give the original size

An example is:

`https://graph.facebook.com/v2.8/1084855054970797/picture?access_token=Your_Access_Token`

Fig. 10 shows a sample response from API.

```
{
  "data": {
    "is_silhouette": false,
    "url": "https://scontent.xx.fbcdn.net/v/t1.0-9/p720x720/16682033_1084855054970797_8061484663955976808_n.jpg?oh=4a13bc07cb22e3d6461b3be33603be00&oe=59063C4E"
  }
}
```

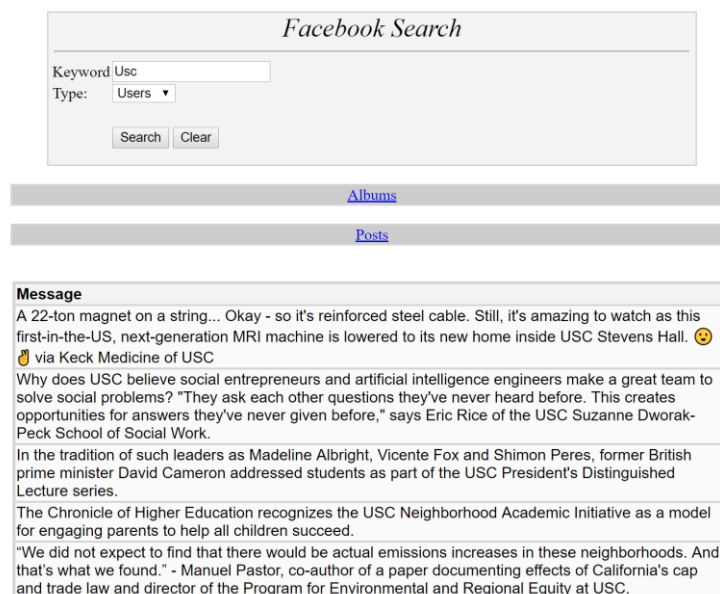
**Figure 10:** A Sample Response of the original High Resolution Image

The ‘url’ value is the actual URL of the high-resolution image.

The “Posts” sub-section should display a maximum of 5 posts (as shown in Fig. 12). Please note that expanding or hiding sub areas should be implemented using JavaScript and you are not allowed to use JQuery.



**Figure 11:** The albums are displayed only when they are clicked. Posts are hidden.



**Figure 12:** When posts are clicked, Albums are hidden.

If the API service returns an empty result set, the page should display “*No Albums have been found*” instead of albums section and “*No Posts have been found*” instead of posts section. A sample output is shown in Fig. 13.

The screenshot shows a web interface titled "Facebook Search". It contains a search form with a "Keyword" input field containing the text "Usc" and a "Type" dropdown menu currently set to "Users". Below these fields are two buttons: "Search" and "Clear". Below the search form, there are two separate light gray rectangular boxes. The first box contains the text "No Albums has been found" and the second box contains the text "No Posts has been found".

**Figure 13:** Search Result When clicking the View Details link of the first result

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

- Based on the input data in the search form, construct a web service URL to retrieve the output from the API service.
- Parse the returned JSON and extract the values.
- Display the user’s albums and posts in tabular format.

## 2.7. Saving Previous Inputs

In addition to displaying the results, the PHP page should maintain the provided values to display the current result. For example, if a user searches for “*Keyword: USC, Type: Events*”, the user should see what was provided in the search form when displaying the results.

When clicking on the “Search” button, the page should display the result retrieved from the *Facebook Graph API* and retain the values provided in the search form. In addition, when clicking on the “Details” link, the page should display the result retrieved from the web service and keep the value 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.

## 3. Hints

### 3.1. HOW TO GET THE FACEBOOK APP ID AND APP SECRET

To use any of the *Facebook Graph APIs* you should first register as a “Facebook developer” at <https://developer.facebook.com/>. After registration, you will be able to create your own Facebook Application ID.



### 3.3. INSTALL FACEBOOK SDK FOR PHP

It is required that you install Facebook SDK for this PHP application. Refer

<https://developers.facebook.com/docs/php/gettingstarted>.

Go with the manual method.

1. Download php-graph-sdk-5.0.0.zip
2. Extract it
3. Upload it under the directory which hosts your PHP file
4. In your code file include the line

**require\_once \_\_DIR\_\_ . '/php-graph-sdk-5.0.0/src/Facebook/autoload.php';**

### 3.4. CONSTRUCTING THE URL FOR FACEBOOK GRAPH API CALLS

The Facebook SDK for PHP is a library with powerful features that enable PHP developers to easily integrate Facebook login and make requests to the Graph API. See:

<https://developers.facebook.com/docs/php/gettingstarted>

Once you obtain the app access token, you can use the APIs and get the results. To know more about the fields and parameters for the APIs, please visit:

<https://developers.facebook.com/docs/graph-api/using-graph-api/>

For reference, you can visit the Graph API Explorer at:

<https://developers.facebook.com/tools/explorer>

In the explorer, you can directly get code by clicking “generate code” and use that in order to make API calls.

### 3.5 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](https://console.developers.google.com/flows/enableapi?apiid=geocoding_backend&keyType=SERVER_SIDE&reusekey=true)).
- Create or select a project.
- Click Continue to Enable the API.
- Go to Credentials to get a Server key (and set the API Credentials).

### 3.6. 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>.

To read the contents of a JSON-formatted object, you can use the “*file\_get\_contents*” function.

## 4. Files to Submit

In your course homework page, you should update the **HW6 link** to refer to your new initial web page for this exercise. Also, submit your file (A single .php file named search.php) electronically to the csci571 account so that they 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 Facebook APIs directly from JavaScript**, bypassing the Apache proxy. Doing this will result in an **8-point penalty.**
- **The link to the video is :** <https://youtu.be/tWu2b3324Bk>