

Homework 6: Search Server-side Scripting using Python Flask, JSON, and Ticketmaster API

1. Objectives

- Get experience with the Python programming language and Flask framework
- Get experience with the Google API and Ticketmaster API
- Get experience creating web pages using HTML, CSS, JavaScript, DOM, JSON format and XMLHttpRequest object
- Get experience using JSON parsers in Python and JavaScript
- Getting hands-on experience in Google Cloud App Engine.

1.1. Cloud exercise

The back end of this homework must be implemented in the cloud on Google Cloud App Engine, using Python.

- See assignment #5 for installation of needed components on Google Cloud.
- See the hints in section 3; a lot of reference material is provided to you.
- For Python and Flask kick-start, please refer to the Lecture slides on the class website.
- You must refer to the grading guidelines, the video, the specs, and Piazza. Styling will be graded, and the point's breakup is mentioned in the grading guidelines.

2. Description

In this exercise, you are asked to create a webpage that allows you to search for events information using the [Ticketmaster API](#), and the results will be displayed in a tabular format. The page will also provide event details.

2.1. Description of the Search Form

The user first opens a web page (for example, **event.html**, or any valid web page name). You should use the [ipinfo.io 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 the event he/she wants to search (categories include Music, Sports, Arts & Theatre, Film, Miscellaneous) from a drop-down list. The default value for the "Category" drop-down list is "Default", which covers all the "types" provided by the *Ticketmaster 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 [ipinfo.io 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.

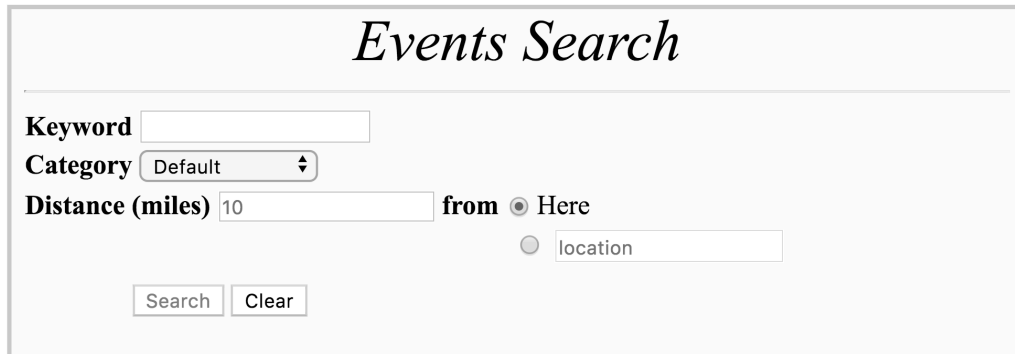


Figure 1(a): Initial Search Screen (Search button disabled)

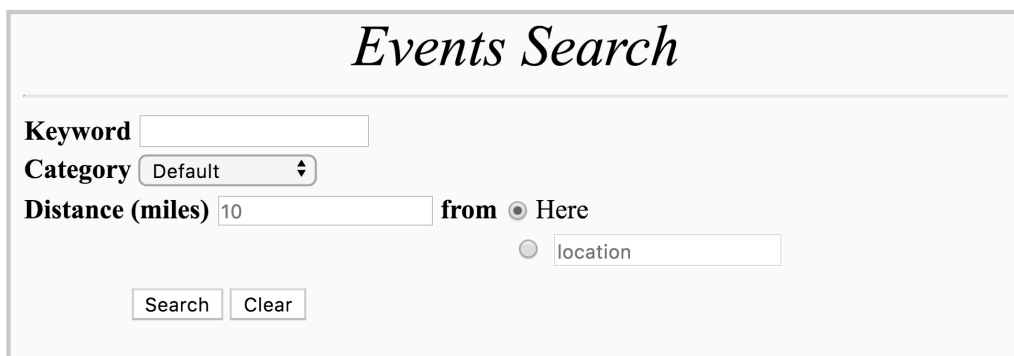


Figure 1(b): Search Screen (after fetched location)

The search form has two buttons:

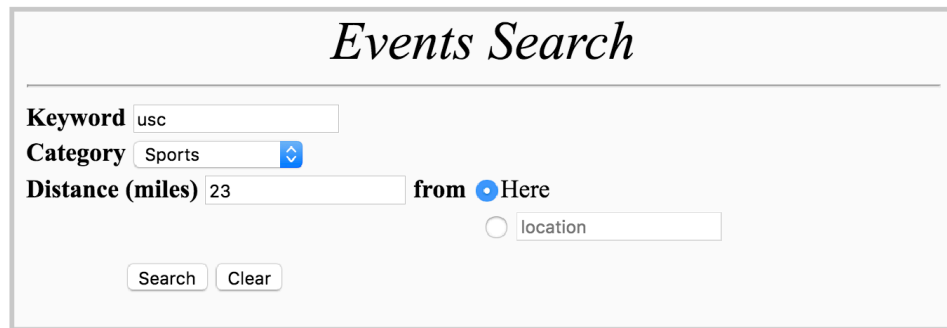
- **Search** button: The button must be disabled while the page is fetching the user’s geolocation and must be enabled once the geolocation is obtained. An example of valid input is shown in Figure 2. Once the user has provided valid input, your client script should send a request to your web server script with the form inputs. You must use GET to transfer the form data to your web server (do not use POST, as you would be unable to provide a sample link to your cloud services). A Python script using Flask will retrieve the form inputs and send it to the *Ticketmaster API* event search service. You need to use the *Flask* Python framework to make all the API calls.

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.

Using XMLHttpRequest or any other JavaScript calls for anything other than calling your own “cloud” backend will lead to a 4-point penalty. Do not call the Ticketmaster API directly from JavaScript.

Define routing endpoints and make your API call from the Python backend. The recommended tutorial for *Flask* and more importantly, routing, can be found at the following link: <https://flask.palletsprojects.com/en/1.1.x/>

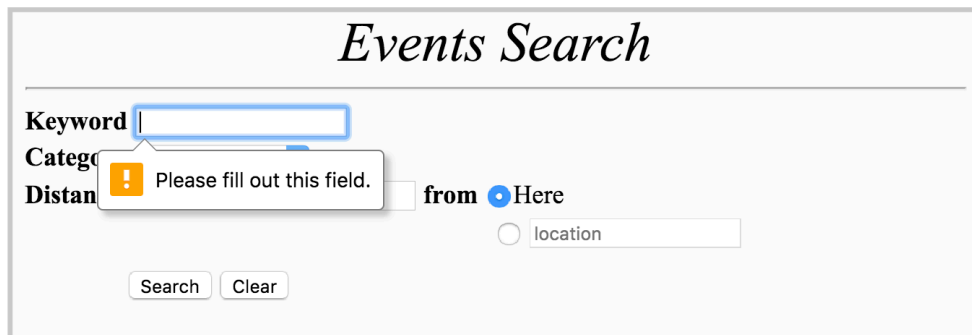
- **Clear** button: This button must clear the result area (below the search area) and set all fields to the default values in the search area. The clear operation must be done using a JavaScript function.



The image shows a web form titled "Events Search". It contains the following fields and controls:

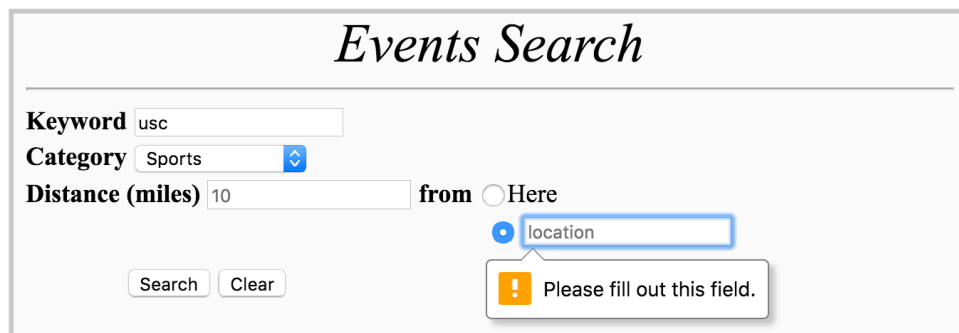
- Keyword:** A text input field containing the value "usc".
- Category:** A dropdown menu with "Sports" selected.
- Distance (miles):** A text input field containing the value "23".
- from:** A radio button group with "Here" selected and "location" unselected.
- Search and Clear buttons:** Two buttons at the bottom of the form.

Figure 2: An Example of a valid Search



The image shows the "Events Search" form with the "Keyword" field empty. A blue border highlights the empty field, and a tooltip with an exclamation mark icon and the text "Please fill out this field." points to it. The other fields are in their default states: "Category" is "Sports", "Distance (miles)" is empty, and "from" is "Here".

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



The image shows the "Events Search" form with the "location" radio button selected. A blue border highlights the "location" text, and a tooltip with an exclamation mark icon and the text "Please fill out this field." points to it. The other fields are in their default states: "Keyword" is "usc", "Category" is "Sports", and "Distance (miles)" is "10".

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

2.2 Displaying Events Results Table

In this section, we outline how to use the form inputs to construct the calls to the RESTful web services to the *Ticketmaster API* service and display the result in the web page.

The *Ticketmaster API* is documented here:

<https://developer.ticketmaster.com>

If the location edit box is selected, your client JavaScript 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).

2.2.1 Geocoding

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.

```

▼ 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

The latitude and longitude of the address are needed when constructing a restful web service URL to retrieve all entities matching the user query. The *Ticketmaster API* “Event Search” service uses *geohash* to represent the address location, instead of *latitude* and *longitude*. You can convert latitude/longitude to geohash on the client or server side.

A Python library for geohash encoding is available online. The source code can be found here:

<https://pypi.org/project/geolib/>

Call the function *geihash.encode()* as in:

```
geohash = geohash.encode('70.2995', '-27.9993', 7)
```

A JavaScript library for the same lat/long -> geohash conversion is available here:

<https://www.movable-type.co.uk/scripts/geohash.html>

and here:

<https://github.com/chrisveness/latlon-geohash/blob/master/latlon-geohash.js>

Call the function *Geohash.encode()* as in:

```
const geohash = Geohash.encode(52.205, 0.119, 7);
```

2.2.2. Ticketmaster API Event Search service

The *Ticketmaster API* Event Search service is documented here:

<https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/#search-events-v2>

The *Ticketmaster API* Event Search service expects the following parameters:

- **apikey:** Your application's API key. This key identifies your application for purposes of quota management.
- **geoPoint:** The *geohash* around which to retrieve event information. The geohash is calculated by latitude and longitude values.
- **radius:** The distance within which to return event results.
- **segmentId:** Filters the results to events matching the specified type id. Only one category may be specified (see Table 1). Leave the field empty means searching in all categories.

Table 1: Ticketmaster API - SegmentId Values for Various Event Categories

<i>Category</i>	<i>SegmentId</i>
Music	KZFzniwnSyZfZ7v7nJ
Sports	KZFzniwnSyZfZ7v7nE
Arts & Theatre	KZFzniwnSyZfZ7v7na
Film	KZFzniwnSyZfZ7v7nn
Miscellaneous	KZFzniwnSyZfZ7v7nl

- **unit:** Unit of the radius. There are two options, “miles” and “km”. Use “miles”.
- **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 *Ticketmaster API* Event Search that searches for the nearby sport events near the University of Southern California within a 10 miles radius is shown below:

```
https://app.ticketmaster.com/discovery/v2/events.json?apikey= YOUR_API_KEY
&keyword=University+of+Southern+California&segmentId=KZFzniwnSyZfZ7v7nE
&radius=10&unit=miles&geoPoint=9q5cs
```

Figure 5 shows an example of the JSON response returned by the *Ticketmaster API* Event Search service response.

```

▼ events:
  ▼ 0:
    ▶ name: "Colorado Buffaloes Footb...at USC Trojans Football"
    ▶ type: "event"
    ▶ id: "Z7r9jZ1AeaM_4"
    ▶ test: false
    ▶ url: "http://www.ticketnow.co...ketList.aspx?PID=2277655"
    ▶ locale: "en-us"
    ▶ images: [...]
    ▶ distance: 2.31
    ▶ units: "MILES"
    ▶ sales: {}
    ▶ dates: {}
    ▶ classifications: {}
    ▶ outlets: {}
    ▶ seatmap: {}
    ▶ _links: {}
    ▶ _embedded: {}
  ▼ 1:
    ▶ name: "UCLA Bruins Men's Soccer...aska-Omaha Men's Soccer"
    ▶ type: "event"
    ▶ id: "vvG1iZ4251I7uZ"
    ▶ test: false
    ▶ url: "https://www.ticketmaster...8/event/0B0054F2CAA04340"
    ▶ locale: "en-us"
    ▶ images: [...]
    ▶ distance: 9.88
    ▶ units: "MILES"
    ▶ sales: {}

```

Figure 5: A sample JSON response returned by the *Ticketmaster API* Event Search

The Python script 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 a **JSON-formatted object**. You should use JavaScript to parse the JSON object, extract the needed fields, and display the results in a tabular format. A sample output is shown in Figure 6. The displayed table includes five columns: Date, Icon, Event Name, Genre, and Venue Name. If the API service returns an empty result set, the page should display “No records have been found” as shown in Figure 7.

Events Search

Keyword
 Category
 Distance (miles) from ☒ Here ☐ location










Date	Icon	Event	Genre	Venue
2018-12-30 13:25:00		Los Angeles Rams vs. San Francisco 49ers	Sports	Los Angeles Memorial Coliseum
2018-10-28 13:25:00		Los Angeles Rams vs. Green Bay Packers	Sports	Los Angeles Memorial Coliseum
2018-09-23 13:05:00		Los Angeles Rams vs. Los Angeles Chargers	Sports	Los Angeles Memorial Coliseum
2018-09-27 17:20:00		Los Angeles Rams vs. Minnesota Vikings	Sports	Los Angeles Memorial Coliseum
2018-09-16 13:05:00		Los Angeles Rams vs. Arizona Cardinals	Sports	Los Angeles Memorial Coliseum
2018-11-11 13:25:00		Los Angeles Rams vs. Seattle Seahawks	Sports	Los Angeles Memorial Coliseum
2018-12-16 17:20:00		Los Angeles Rams vs. Philadelphia Eagles	Sports	Los Angeles Memorial Coliseum
2018-12-31		Los Angeles Rams VIP Packages	Sports	Los Angeles Memorial Coliseum
2018-11-19		Los Angeles Rams VIP Packages (Mexico City Game)	Sports	Los Angeles Memorial Coliseum

Figure 6: An Example of a Valid Search result

Events Search

Keyword

Category Default

Distance (miles)
from ☒ Here
 ☐

No Records has been found

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 2.

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

HTML Table Column	API service response
Date	The value of the “ <i>localDate</i> ” and “ <i>localTime</i> ” attributes that is part of “ <i>events</i> ” object
Icon	The value of the “ <i>images</i> ” attribute that is part of the “ <i>events</i> ” object.
Event	The value of the “ <i>name</i> ” attribute that is part of the “ <i>events</i> ” object.
Genre	The value of the “ <i>segment</i> ” attributes that is part of the “ <i>events</i> ” object.
Venue	The value of the “ <i>name</i> ” attribute that is part of the “ <i>venue</i> ” object inside “ <i>events</i> ” object.

2.3 Displaying Event Details (Event details)

In the search result table, if the user clicks on the name of an event, the page should request the detailed information using the *Event Details API* and direct to a section populated with this information, documented at:

<https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/#event-details-v2>

To retrieve event details, the request needs two parameters (output should be JSON):

- **id:** ID of the event
- **apikey:** Your application's API key. This key identifies your application for purposes of quota management.

2.3.1 Event Details

An example of an HTTP request to the *Event Details API* is shown below:

```
https://app.ticketmaster.com/discovery/v2/events/{id}?apikey=YOUR_API_KEY&
```

Figure 8 shows a sample response.

```
name: "Los Angeles Rams vs. San Francisco 49ers"
type: "event"
id: "vvG1IZ4kDLAPsu"
test: false
url: "https://www.ticketmaster.com/los-angeles-rams-vs-san-francisco-los-angeles-califor"
locale: "en-us"
images: [...]
sales: {}
dates:
  start:
    localDate: "2018-12-30"
    localTime: "13:25:00"
    dateTime: "2018-12-30T21:25:00Z"
    dateTBD: false
    dateTBA: false
    timeTBA: false
    noSpecificTime: false
    timezone: "America/Los_Angeles"
  status: {}
  spanMultipleDays: false
classifications: [...]
promoter: {}
promoters: [...]
priceRanges: [...]
seatmap:
  staticUrl: "https://s1.ticketm.net/t...enue/maps/wes/70057s.gif"
_links: {}
```

Figure 8: An example of a team photo search response (Keyword: Rams)

The Python script 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-formatted object**. You should use JavaScript to parse the JSON object and display the results in a similar format as Figure 9. When clicking on the artist's name (or team name), a page with artist's upcoming events will open in a new tab. When clicking on the "Ticketmaster" link, under "Buy Ticket At", a page to buy tickets online will open in a new page. If the returned JSON stream doesn't contain certain fields, those fields will not appear on the detail page. A sample output is shown in **Figure 9**. Figure 9(a) shows a result with all fields, Figure 9(b) shows a result with missing fields such as "artist", "genre", "price Range", and "seat map".

Los Angeles Rams vs. San Francisco 49ers

Date

2018-12-30 13:25:00

Artist / Team

Los Angeles Rams | San Francisco 49ers

Venue

Los Angeles Memorial Coliseum

Genres

NFL | Football | Sports | Team | Group

Price Ranges

62 - 275 USD

Ticket Status

Onsale

Buy Ticket At:

Ticketmaster



Figure 9(a): An Example of a Valid Search result

Jen's Test Site

Date

2019-09-30 08:45:00

Venue

NOS Events Center

Ticket Status

Onsale

Buy Ticket At:

Ticketmaster

Figure 9(b): An Example of a Valid Search result with missing fields

When the search result contains at least one field, you need to map the data extracted from the API result to render the HTML result table as described in Table 3.

Table 3: Mapping the result from *Event Details API* into HTML Table

HTML Key	API service response
Date	The value of the “ <i>localDate</i> ” and “ <i>localTime</i> ” attributes that is part of the “ <i>dates</i> ” object
Artist/Team	The value of the “ <i>name</i> ” attribute that is part of the “ <i>attractions</i> ” object, segmented by “ ”
Venue	The value of the “ <i>name</i> ” attribute that is part of the “ <i>venue</i> ” object.
Genre	The value of the “ <i>subGenre</i> ”, “ <i>genre</i> ”, “ <i>segment</i> ”, “ <i>subType</i> ”, and “ <i>type</i> ” attributes that is part of the “ <i>classifications</i> ” object, segmented by “ ”
Price Ranges	The value of the “ <i>min</i> ” and “ <i>max</i> ” attributes that are part of

	the “ <i>priceRanges</i> ” object, combined with “-”
Ticket Status	The value of the “ <i>status</i> ” attribute that is part of the “ <i>dates</i> ” object.
Buy Ticket At	The value of the “ <i>URL</i> ” attribute.
Seat Map	The value of the “ <i>staticUrl</i> ” attribute that is part of the “ <i>seatmap</i> ” object.

Note that:

- You must use Python to request all JSON objects from Ticketmaster APIs
- You may call Google Geocoding API from client side using JavaScript or from server using Python
- Do not call ipinfo.io API from Python on the server, as you would get the location of the server at Google Cloud instead of the location of the user.
- Expanding or hiding sub-areas should be implemented using DOM.

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 Ticketmaster *API* service.
- Pass the (possibly edited) JSON to the client side and parse JSON using JavaScript.
- Display the events information in the proper format.

3. Hints

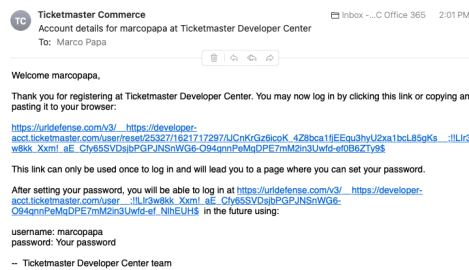
3.1 How to get Ticketmaster API Key

- To get a Ticketmaster API key, please follow these steps:
- Create a new account at:

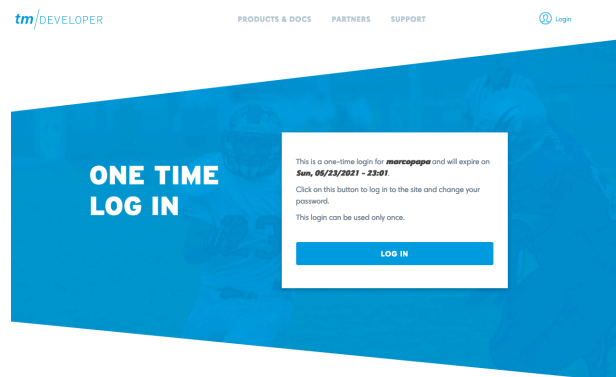
<https://developer-acct.ticketmaster.com/user/register>

The screenshot shows the 'CREATE NEW ACCOUNT' form on the Ticketmaster Developer portal. The form includes fields for First Name, Last Name, Company Name, Company Site URL, Phone Number, Application URL, Username, and E-mail address. There is a checkbox for 'I agree to the Terms of Use' and a 'CREATE NEW ACCOUNT' button. The background is blue with a white form area.

Enter your data and click **CREATE NEW ACCOUNT**. You should receive an e-mail with a link.



Click on the link in the email. You'll be redirected to the one-time-login. Click **LOG IN**.



You'll be redirected to your Profile page where you can select your password.

EDIT PROFILE

A screenshot of the "EDIT PROFILE" page. The page is divided into two main sections: "Personal Information" and "Change Password". The "Personal Information" section contains fields for "First Name", "Last Name", "Username", "Phone Number", and "E-mail address". The "Change Password" section contains fields for "New password" and "Confirm password". The "New password" field is highlighted with a red box. Below the "New password" field, there is a "Password strength" indicator showing a green checkmark. The "Confirm password" field also has a green checkmark. The page also includes a "Manage your OpenIDs" link in the top right corner.

Select the appropriate time zone, America/Los Angeles.

Time zone

America/Los Angeles: Saturday, May 22, 2021 - 14:02 -0700

☒ I Agree to the Terms and Conditions *

SAVE

DELETE ACCOUNT

- Once logged in, click your name on the right top corner and select “My Apps”.
- You should see an “approved” app. Expand the app info. You should see a *Consumer Key*. That is the key to use with the Ticketmaster APIs.

The screenshot shows the "Keys" tab selected in the top navigation bar. Below the tabs, there's a note stating that keys are used to access API products and may need approval. A red box highlights the "Consumer Key" field, which contains a long alphanumeric string. Other fields shown include "Consumer Secret", "Key Issued" date, "Key Expires" status, "Product" name, and "Public APIs" usage limit.

Field	Value
Consumer Key	IBvD3j... [redacted]
Consumer Secret	CpChwOuU5ggnolic
Key Issued	Sat, 05/22/2021 - 14:01
Key Expires	Never
Product	Status
Public APIs	Approved 800 requests every 1 day
OAuth Product	Approved 100 requests every 1 minute

3.2 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 the 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. Preferably use a Gmail account.

3.3 Get IPInfo.io API Key

- Go to <https://ipinfo.io/> and sign up for free
- A token would be provided after successful sign up

An example call is as follows:

`https://ipinfo.io/?token=YOUR_TOKEN_ID`

The following article introduces some similar APIs, so you have more choices for your homework assignment #6, if you wish to try out different APIs:

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

Use of Freegeoip API is not recommended.

3.4 Deploy Python file to the cloud GAE

You should use the domain name of the GAE service you created in Assignment #5 to make the request. For example, if your GAE server domain is called **example.appspot.com**, the following links will be generated:

`http://example.appspot.com/index.html`

The *example* subdomain in the above URLs will be replaced by your choice of subdomain from the cloud service. You may also use a different page than `index.html`.

Files to deploy:

- client-side files (HTML+CSS+JS)
- server-side file (`main.py`), `.yaml`, `requirements.txt`

The project structure should be similar to the following one:

```
C:.\
|
|  .gcloudignore
|  app.yaml
|  main.py
|  requirements.txt
|
|  --static
|  |
|  |  event.html
|  |
|  --__pycache__
|  |
|  |  app.cpython-37.pyc
```

3.5 Parsing JSON-formatted data in Python

Information on how to parse JSON-formatted data in Python is available here:

<https://docs.python.org/3/library/json.html>

If you use your cloud server as a “proxy” pass-through, you do not have to decode and encode the JSON.

4. Files to Submit

In your course homework page, you should update the **Homework 6** link to refer to your new initial web search page for this exercise (for example, **event.html**). Your files must be hosted on GAE cloud service. Graders will verify that this link is indeed pointing to GAE.

Also, submit your source code file to DEN D2L. **The timestamp will be used to verify if you used any “grace days.”**

****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 can use jQuery for Homework 6 but it is not required.
- You **should not call the Ticketmaster APIs directly from JavaScript**, bypassing the Python proxy. Implementing any one of them in JavaScript instead of Python will result in a **4-point penalty**. Other APIs can be called from JavaScript.