HOMEWORK ASSIGNMENT 6

CSCI 571 – Spring 2023

Abstract

Server-side Scripting using Python, Flask, JSON, AJAX, and the Ticketmaster API

Marco Papa

papa@usc.edu

Homework 6: Server-side Scripting using Python, Flask, JSON, AJAX, and the 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 the XMLHttpRequest object.
- Get experience using JSON parsers in Python and JavaScript.
- Getting hands-on experience in GCP, AWS or Azure.

1.1 Cloud Exercise

The backend of this homework must be implemented in the cloud on Google Cloud App Engine, AWS or Azure, using Python.

- See assignment #5 for the installation of needed components on GCP, AWS or Azure.
- 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 event information using the Ticketmaster API, and the results will be displayed in a tabular format. The page will also provide event and venue details.

2.1. Description of the Search Form

The user first opens a web page (for example, **events.html,** or any valid web page name). The form has 5 components Keyword, Distance (miles), Category, Location, and a checkbox to auto-detect location. You should use the *ipinfo.io API (*See hint 3.3) to fetch the user's geolocation if the location checkbox is checked else the user must enter a location to search. Keyword and Location being text boxes, Distance being numerical field, while Category being a dropdown.

The categories to include are:

- Music
- Sports
- Arts
- Theatre
- Film

- Miscellaneous

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 the current location returned from ipinfo.io API or the location listed in the custom location input box. When the auto-detect location checkbox is selected, the custom location input box must disappear and when it is unchecked, the custom location input box is a required field, and a location string must be entered. The default distance is 10 miles from the chosen location. Use an HTML5 “placeholder” to show “10” in the Distance edit box as the initial value.

An example is shown in **Figure 1**.

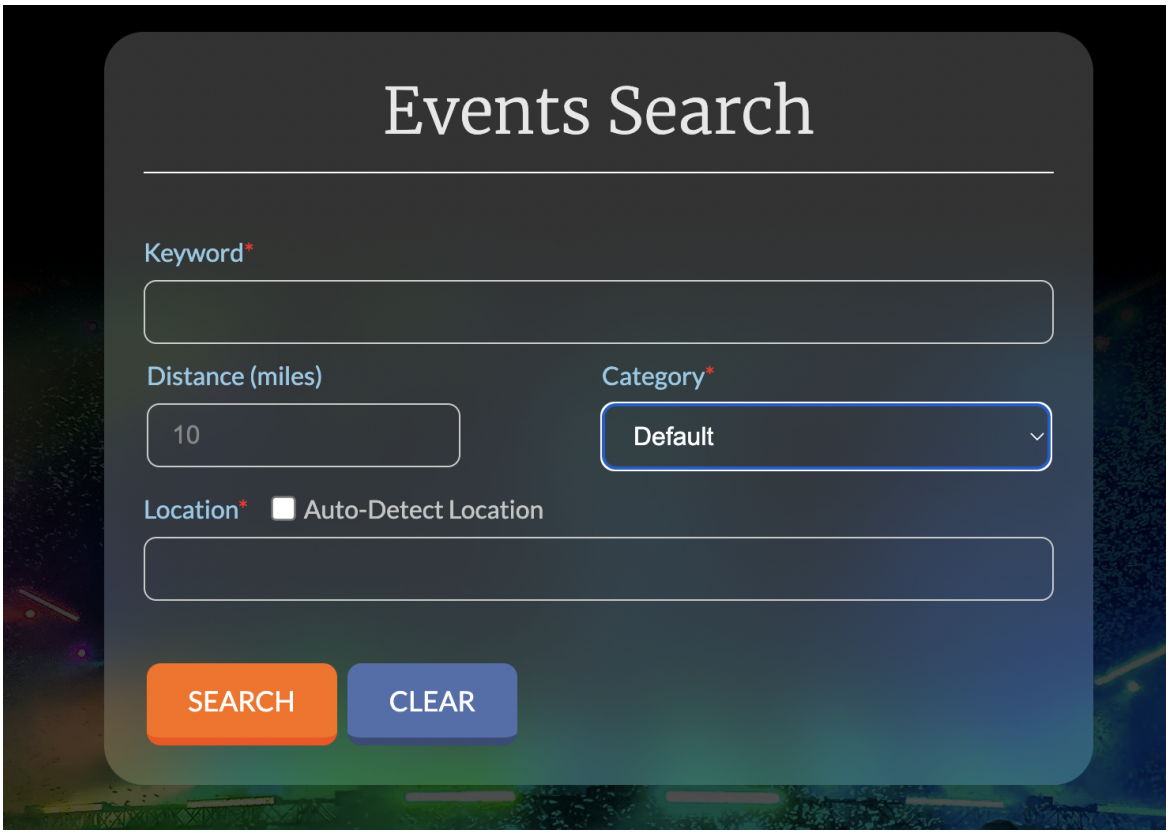


Figure 1: Initial Search Form

The search form has two buttons:

- **SEARCH** button. Selecting this button performs a search using the given events keyword, the distance filter from the given location and the Category of event. An example of valid input is shown in **Figure 2a**. Once the user has provided valid input, your client JavaScript should send a request to your web server Python 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 extract the form inputs and use them to invoke the *Ticketmaster API* event search service. You need to use the *Flask* Python framework to make all the API calls. You may also use Django, though we can only provide limited support on Piazza for Django.

If the user clicks on the SEARCH button without providing a value in the “Keyword” and “Location”(Auto-detect location unchecked) fields , you should show an error “tooltip” that indicates the field is missing. An example is shown in **Figure 2b**. If no values are provided to the “Distance” and “Category” field, it should be taken as 10, and Default respectively.

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. You may use XMLHttpRequest (or equivalent file fetch()) to call the *iPInfo API* and the *Google Maps Geocoding API* service directly from JavaScript (see later).**

Define the 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 the form fields to the default values in the search area. The **CLEAR** operation must be done using a JavaScript function.

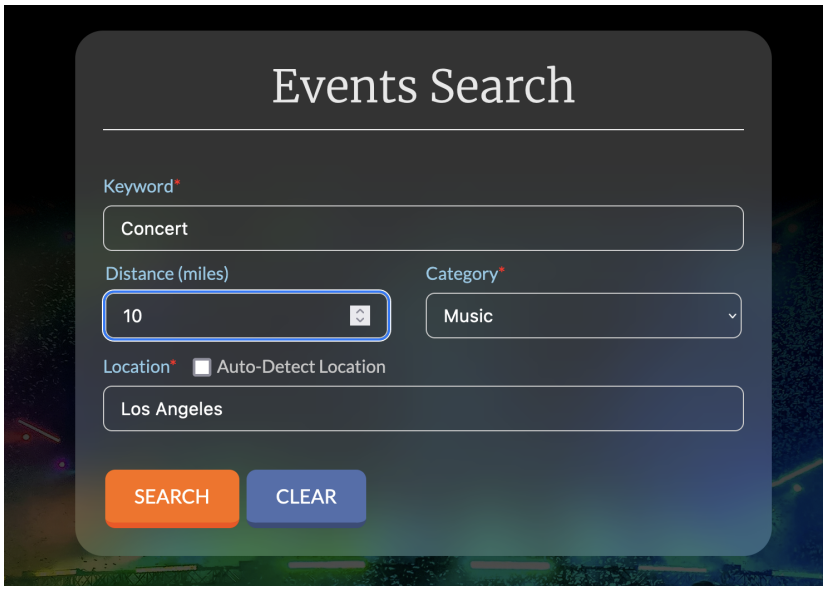


Figure 2a: An Example of a Valid Search

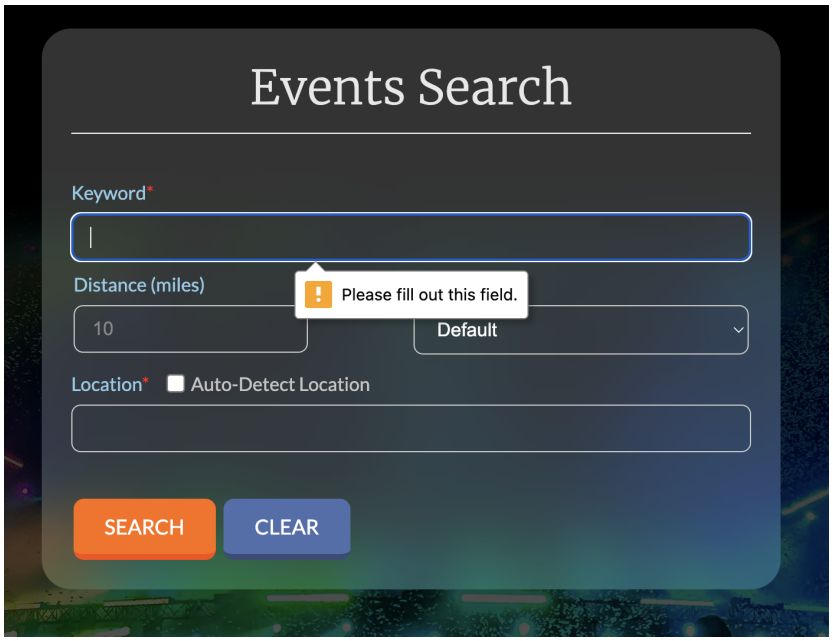


Figure 2b: An Example of an Invalid Search

2.2 Displaying Events Results

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 results in the web page.

The *Ticketmaster API* is documented here:

https://developer.ticketmaster.com

If the **Location** information is used to get events results, your client JavaScript should use the input address to get the geocoding via the *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 location that you want to geocode, in the format used by the national postal service of the country concerned. Additional address elements such as event 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 3.2).

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 3: Example of object from Google Maps Geocoding

Figure 3 shows an example of the JSON object returned in the *Google Maps Geocoding API* web service response.

The latitude (lat) and longitude (lng) of the location are used when constructing a RESTful web service URL to retrieve matching search results. *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). Leaving the field empty means searching in all categories (Default).

Category	***SegmentId***
Music	KZFzniwnSyZfZ7v7nJ
Sports	KZFzniwnSyZfZ7v7nE
Arts & Theatre	KZFzniwnSyZfZ7v7na
Film	KZFzniwnSyZfZ7v7nn
Miscellaneous	KZFzniwnSyZfZ7v7n1

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

- ***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 4 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.ticketsnow.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 4: 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 5a**. 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 found” as shown in **Figure 5b**.

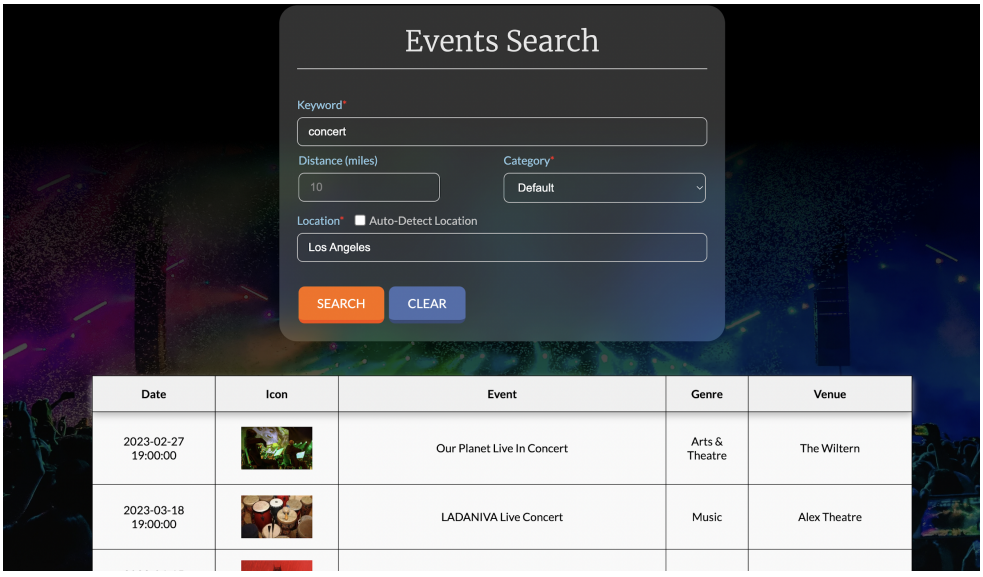


Figure 5a: An Example of a Valid Search result

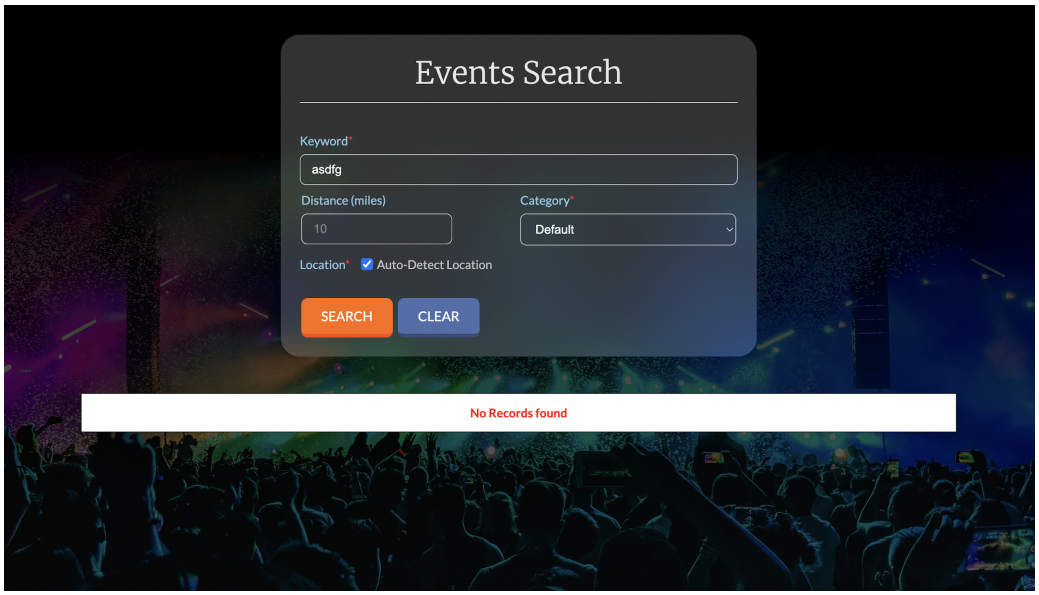


Figure 5b: An Example of an Empty Search result

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**.

HTML Table Column	API service response
Date	The value of the "*localDate*" and "*localTime*" attributes that is part of "*events*" object
Icon	The value of the "*images*" attribute that is part of the "*events*" object.
Event	The value of the "*name*" attribute that is part of the "*events*" object.
Genre	The value of the "segment" attributes that is part of the "*events*" object.
Venue	The value of the "*name*" attribute that is part of the "*venue*" object inside "*events*" object.

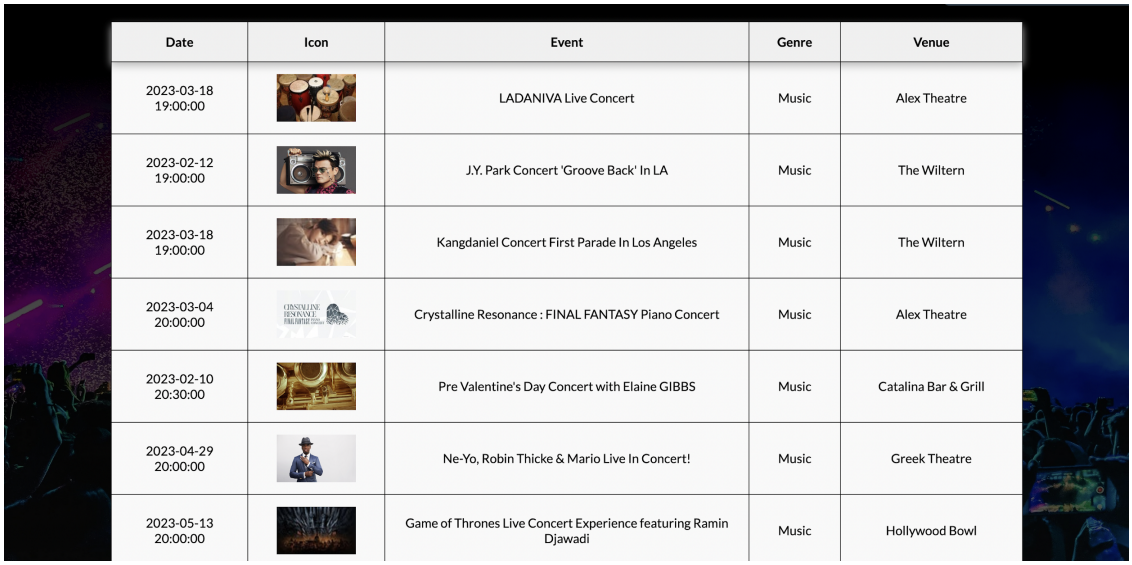
Table 2: Mapping the result from Graph API into HTML table

2.2.3 Displaying table of events

We use the ***Ticketmaster "event search" Service*** to get general information of the event that matches our search criteria (the values on the search form). Please refer to the below link for more information:

https://developer.ticketmaster.com/products-and-docs/apis/getting-started/

The Python script may pass the returned JSON object to the client side “unmodified” (i.e., function as a pass-through”) or parse the returned JSON and extract only the 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 and a table view containing all the events for a given distance, filtered by keyword and categories. A sample output is shown in **Figure 6**. The displayed table includes five columns: **Date, Icon, Event, Genre, and Venue**. The columns may be sorted by clicking on the table header based on Event, Genre, or Venue. For this assignment we will be using the default limit of 20 and fetch the results of 20 events (or less if the matches are less than 20).



Date	Icon	Event	Genre	Venue
2023-03-18 19:00:00		LADANIVA Live Concert	Music	Alex Theatre
2023-02-12 19:00:00		J.Y. Park Concert 'Groove Back' In LA	Music	The Wiltern
2023-03-18 19:00:00		Kangdaniel Concert First Parade In Los Angeles	Music	The Wiltern
2023-03-04 20:00:00		Crystalline Resonance : FINAL FANTASY Piano Concert	Music	Alex Theatre
2023-02-10 20:30:00		Pre Valentine's Day Concert with Elaine GIBBS	Music	Catalina Bar & Grill
2023-04-29 20:00:00		Ne-Yo, Robin Thicke & Mario Live In Concert!	Music	Greek Theatre
2023-05-13 20:00:00		Game of Thrones Live Concert Experience featuring Ramin Djawadi	Music	Hollywood Bowl

Figure 6: Output of Search Results

2.3 Displaying Event Details (Event details and Venue details)

In the search results, if the user clicks on the name of an event, a card should be displayed with detailed results of the event below the table of listed events and the page should automatically scroll all the way down to the card’s location. This can be done by using Javascript functions. The page should request the detailed information using the *Event Details API* and *Venue Search API*, documented at:

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

https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/#search-venues-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 7 shows a sample response.

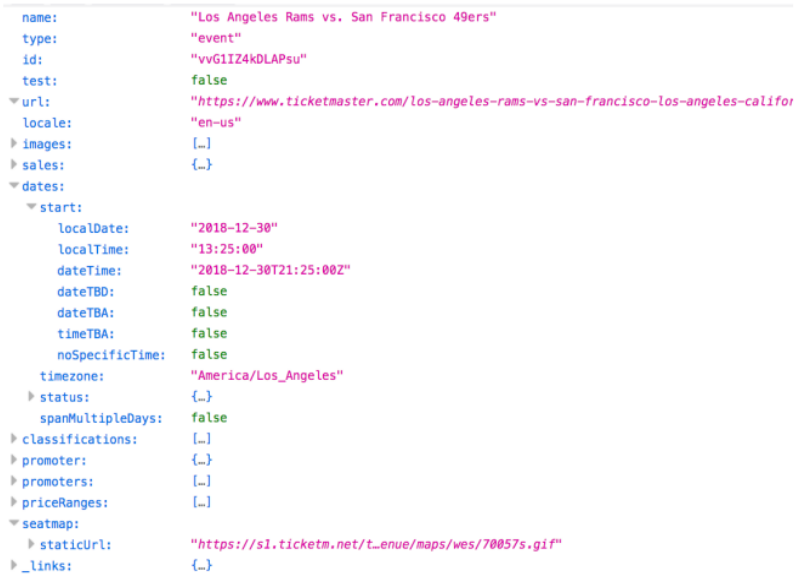


Figure 7: An example of a team photo search response (Keyword: Rams)

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 objects. 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 the 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 **Figures 8 (a-b)**. **Figure 8(a)** shows a result with all fields, **Figure 8(b)** shows a result with missing fields such as "price range".

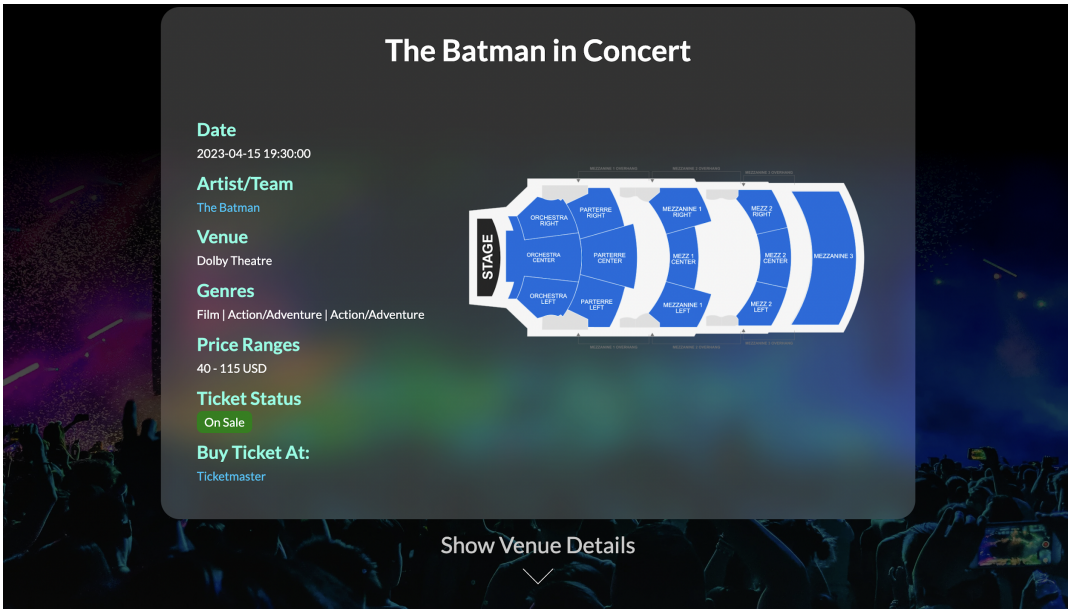


Figure 8(a): An Example of a Valid Search result

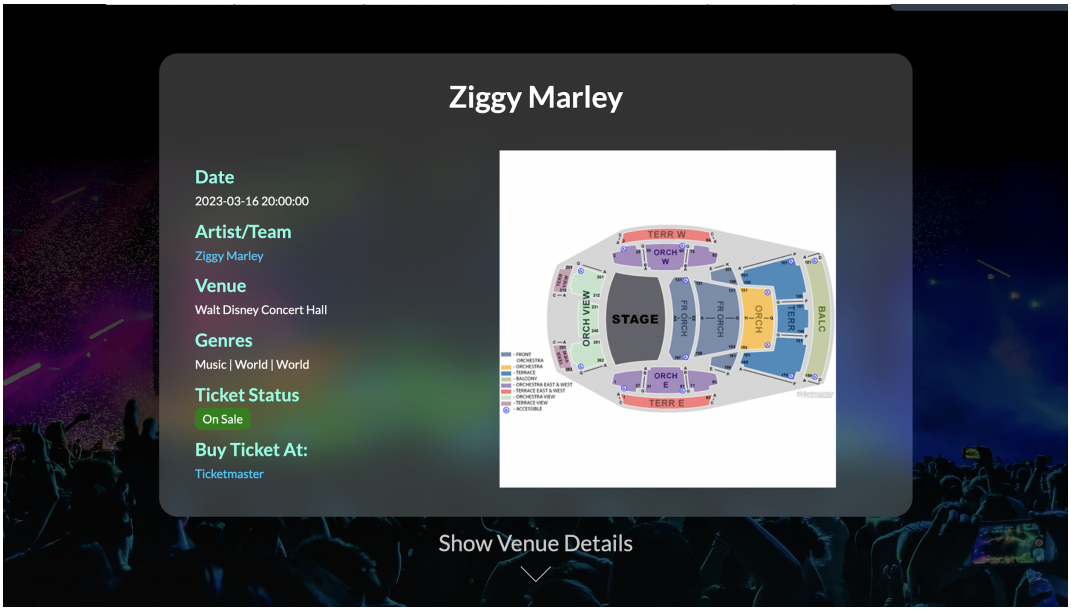


Figure 8(b): An Example of a Valid Search result with missing field (“price range”)

Moreover, the “Ticket Status” in the event details card is color coded. The convention used is follows:

- On sale: Green
- Off sale: Red
- Canceled: Black
- Postponed: Orange
- Rescheduled: Orange

Figure 9 shows the desired appearance to show respective status of the tickets.

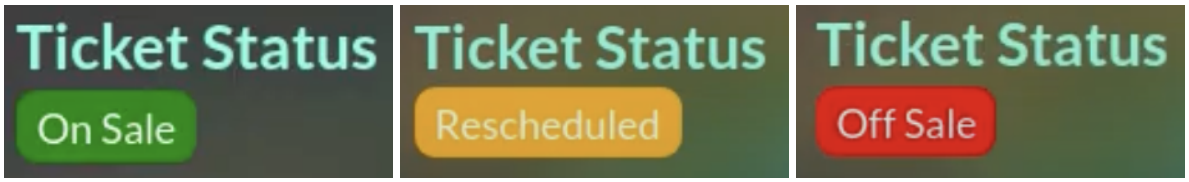


Figure 9: Color coded card for the “Ticket Status” field

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**.

HTML Key	API service response
Date	The value of the “*localDate*” and “*localTime*” attributes that is part of the “*dates*” object
Artist/Team	The value of the *“name”* attribute that is part of the “*attractions*” object, segmented by “ | ”
Venue	The value of the *“name”* attribute that is part of the “*venue*” object.

Genre	The value of the *“subGenre”, “genre”, “segment”, “subType”,* and *“type”* attributes that is part of the “*classifications*” object, segmented by *“ | ”*
Price Ranges	The value of the “*min*” and *“max”* attributes that are part of the “*priceRanges*” object, combined with “ -”
Ticket Status	The value of the “*status.code*” attribute that is part of the “*dates*” object.
Buy Ticket At	The value of the *“URL”* attribute.
Seat Map	The value of the “*staticUrl*” attribute that is part of the *“seatmap”* object.

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

On clicking the down arrow below “Show Venue Details”, a Venue details card should be displayed, the text “Show Venue Details” and the down arrow should disappear. Also the page should automatically scroll downwards to display the venue details card. To retrieve the venue details, the request to *Venue Search API* needs two parameters (output should be `JSON`):

- ***keyword:*** Name of the venue
- ***apikey:*** Your application's API key. This key identifies your application for purposes of quota management.

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

```
https://app.ticketmaster.com/discovery/v2/venues?apikey=YOUR_API_KEY
&keyword=Los%20Angeles%20Memorial%20Coliseum
```

Figure 10 shows a sample response.

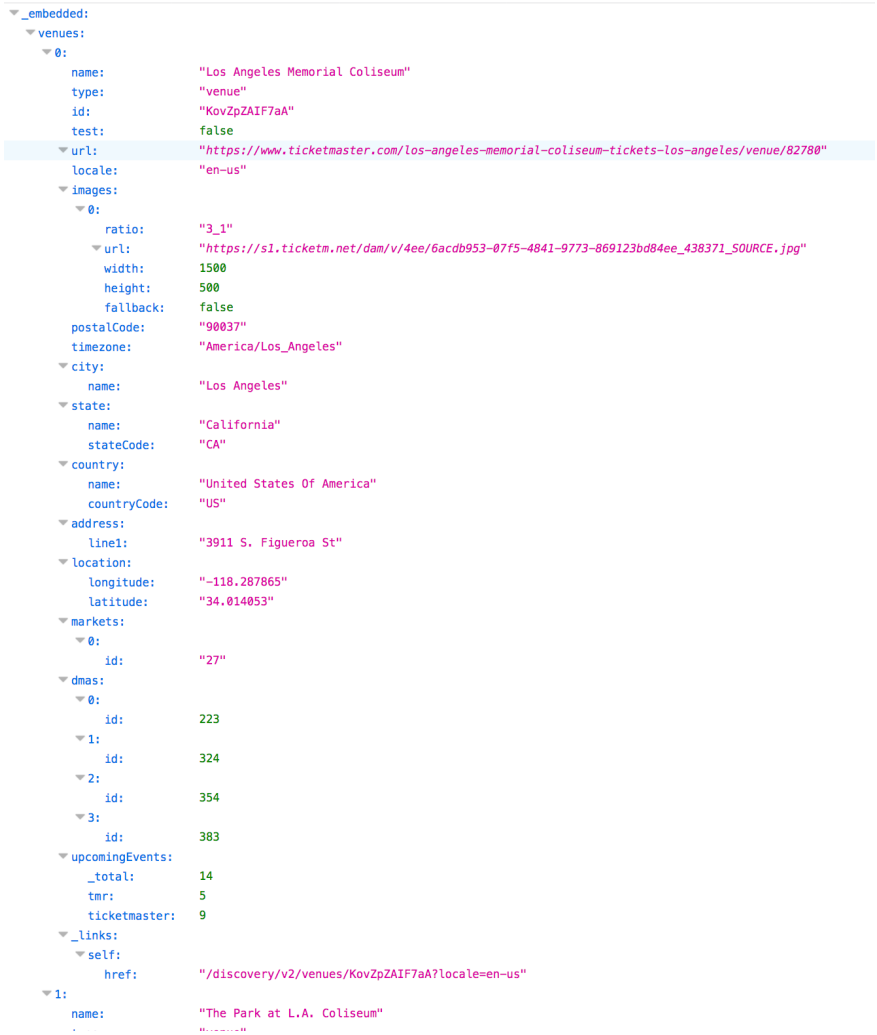


Figure 10: An example of a venue detail result (Keyword: Los Angeles Memorial Coliseum)

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 format**. You should use JavaScript to parse the JSON object and display the results in a similar format to **Figure 11**.

There are two parts in Venue details. The first part displays venue's location information and a hyperlink that will redirect users to location on Google Maps. This can be done by using Google maps URLs (https://developers.google.com/maps/documentation/urls/get-started#search-action). Here the query text will contain the entire address: <Name of venue>, <street address>, <city>, <state>, <zip code>. When clicking on the link of "More events at this venue", a page which displays upcoming events in this venue will open in a new page. If the returned JSON file doesn't contain certain fields, the value of those fields will be set as "N/A". A sample output is shown in **Figure 11**.

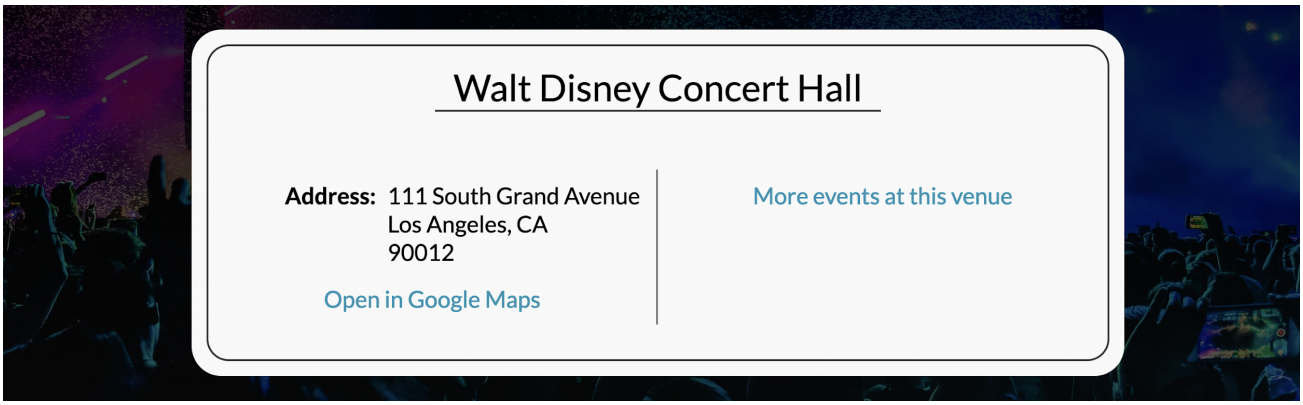


Figure 11: An Example of a Valid Venue Detail

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 4**.

HTML Table Row	API service response
Name	The value of the "*name*" attributes
Address	The value of the *"line1"* attribute that is part of the "*address*" object.
City	The value of the *"name"* attribute of "city" object and "stateCode" attribute of "state" object, connected by a comma.
Postal Code	The value of the "*postalCode*
Upcoming Events	The value of the "*url*" attribute

Table 4: Mapping the result from *Venue Search API* into HTML Table

When the user clicks on the button (see **Figure 12**), the "venue info" sub-section should be expanded (see the video for the behavior).

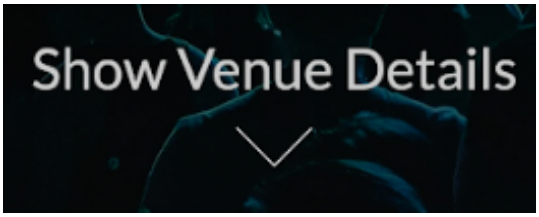


Figure 12: Button to expand Venue details

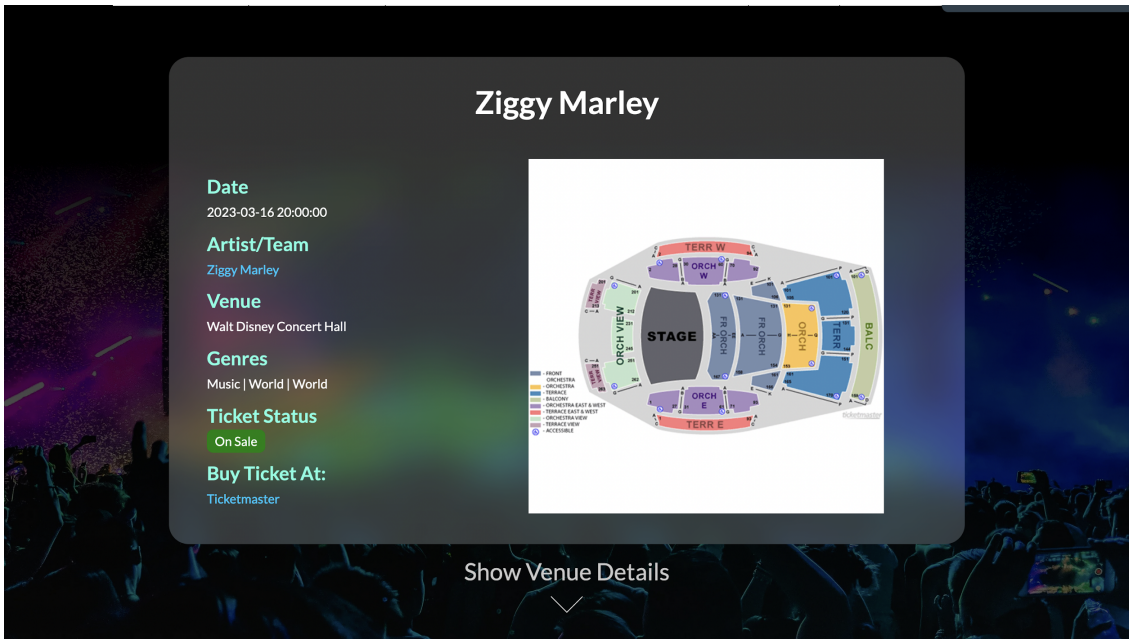


Figure 13a: The Venue Details are hidden.

The "venue info" sub-section should display the venue info, as shown in **Figure 13b**.

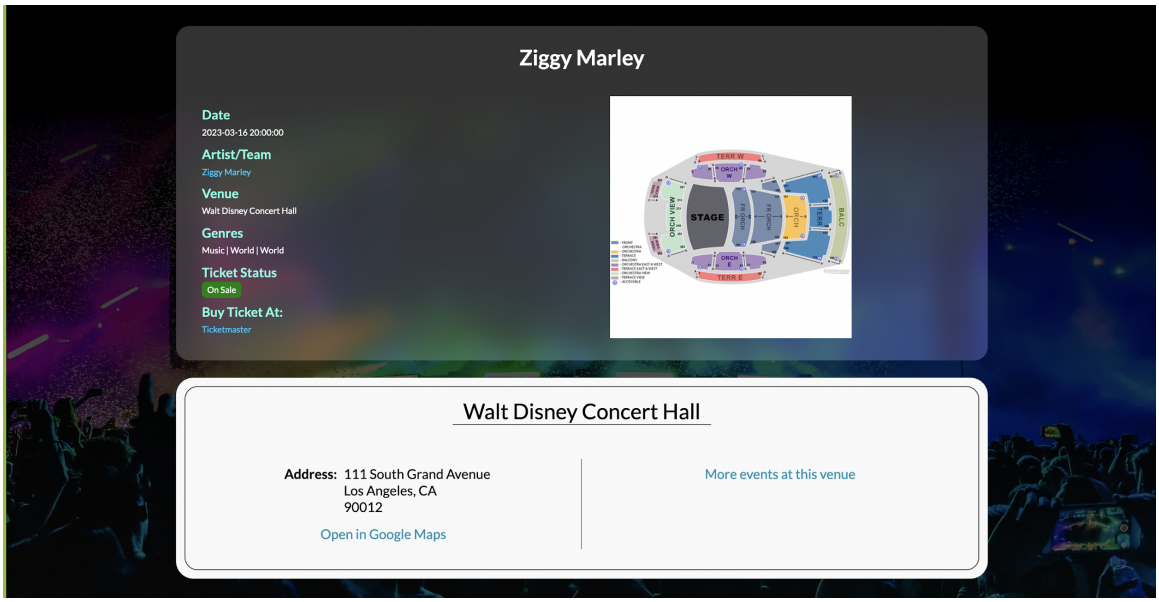


Figure 13b: When venue info is clicked.

Note that:

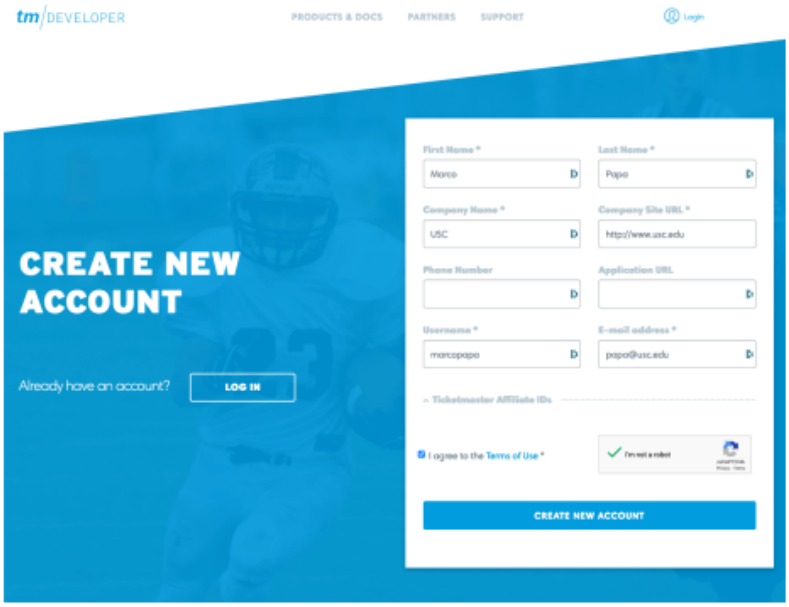
- **You must use Python to request all JSON objects from the Ticketmaster APIs**
- **You may call Google Geocoding API from client side using JavaScript or from server using Python**
- **Call the IPInfo.io API from JavaScript. Do not call ipinfo.io API from Python on the server, as you would get the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

3. Hints

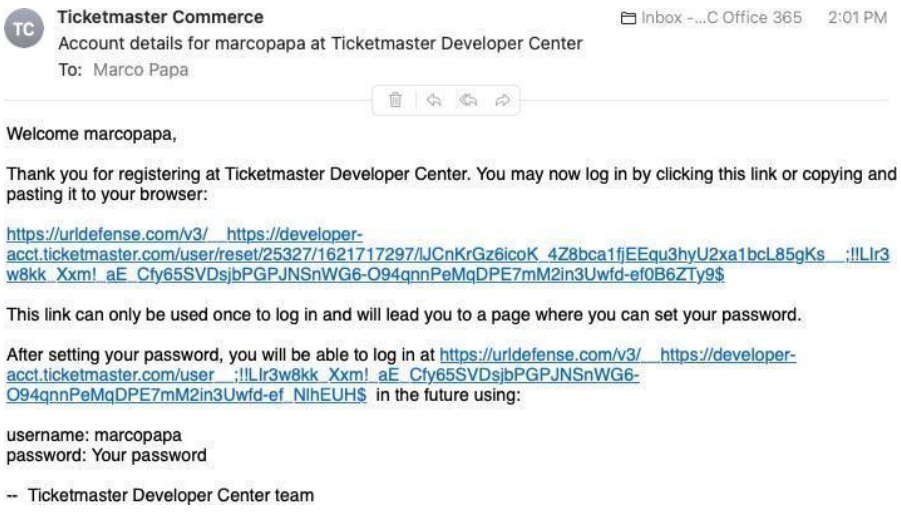
3.1 How to get the Ticketmaster API Key

To get a Ticketmaster API key, please follow these steps:

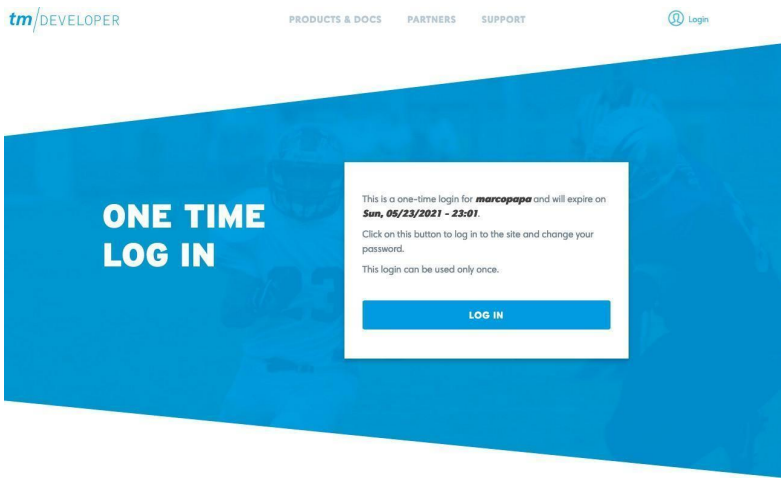
- Create a new account at: https://developer-acct.ticketmaster.com/user/register



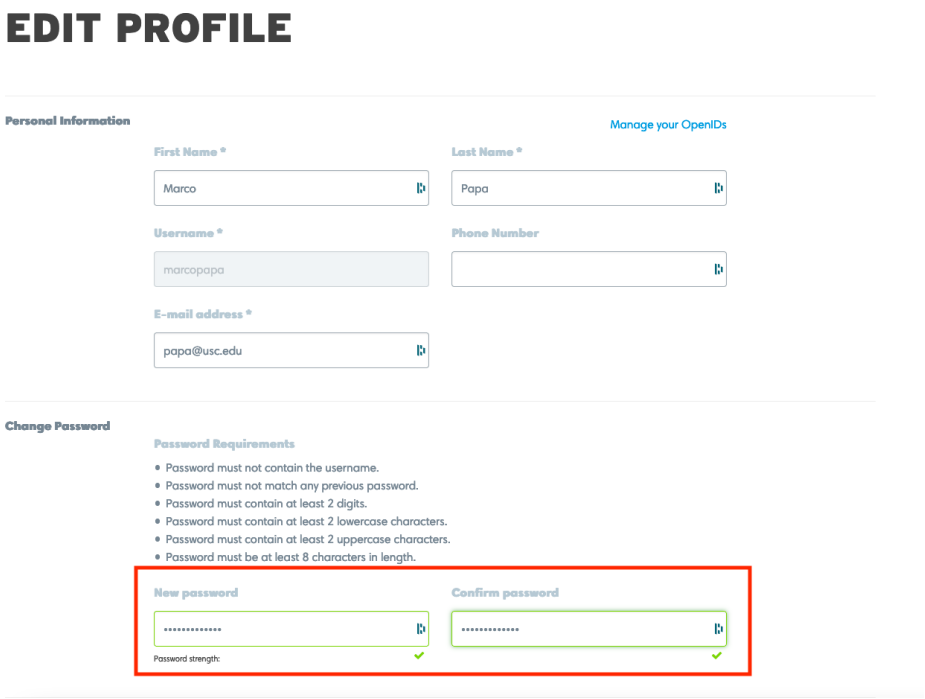
- Enter your data and click CREATE NEW ACCOUNT. You should receive an e-mail with a link.



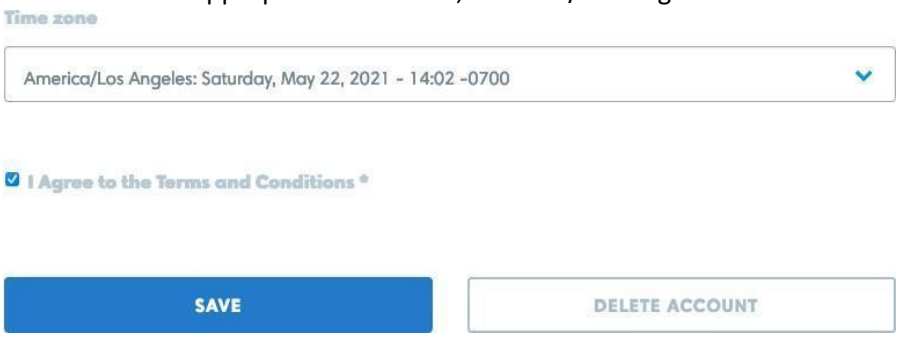
- Click on the link in the email. You'll be redirected to the one-time-login. Click **LOG IN**.



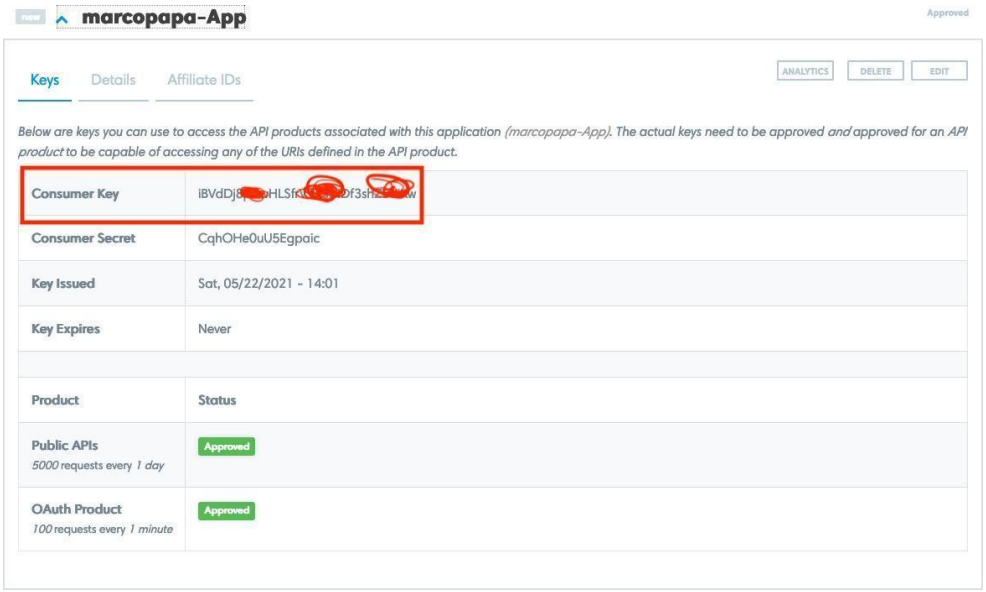
- You'll be redirected to your Profile page where you can select your password.



- Select the appropriate time zone, America/Los Angeles.



- 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.



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 How to 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 (GCP/AWS/Azure)

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

GAE - http://example.appspot.com/index.html

AWS - http://example.elasticbeanstalk.com/index.html

Azure - http://example.azurewebsites.net/index.html

The *example* subdomain in the above URLs will be replaced by your choice of subdomain from the cloud service during setup. You may also use a different page than index.html as your top-level home page.

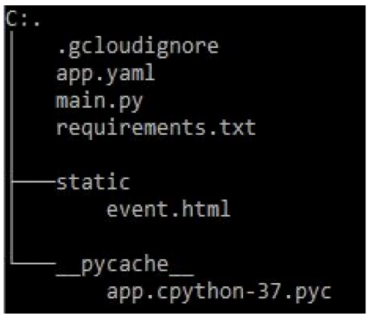
For example, if your GAE server domain is called **example.appspot.com**, the following links will be generated: http://example.appspot.com/index.html

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. For example, the files to deploy on GCP would be:

1. client-side files (HTML+CSS+JS)

2. server-side file (main.py), .yaml, requirements.txt

The project structure may also look like the following one:



You are free to use any folder structure for your project.

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, **events.html**). Your files must be hosted on GAE, AWS, or the Azure cloud service. Graders will verify that this link is indeed pointing to a cloud service.

Also, submit your source code files to DEN D2L. Submit a ZIP file of both front-end and back-end code, in separate folders, plus any additional files needed to build your app (e.g., yaml file). **The timestamp of the ZIP file 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 clarification on Piazza that conflicts with this description and/or the grading guidelines, **Piazza always rules**. In most cases, the clarification is related to an error in the description, or missing information from the description, but **no additional functionality**.
- You can use jQuery for Homework 6, but its use is not required.
- You can use FasS like *Google Cloud Functions*, *Amazon Lambda* and *Azure Functions*, in lieu of building a monolithic application.
- You **should not call any of 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.
- You may call the Google Maps Geocoding API directly from JavaScript.
- **APPEARANCE OF CARD VIEW and TABLE should be as similar as possible to the reference video.** See Reference Video: https://youtu.be/AtfgYB3D3DE