Balloonicorn's After-Party Quest with Ticketmaster

Balloonicorn is hosting a party! Help her build a feature for her web app that will allow guests to search for events on Ticketmaster to keep the party going!

Like many other web services, Ticketmaster supports a public API that third-party developers like us can use to consume and process data.

Learn How to Use Ticketmaster

This section will walk you through the general steps that you'll follow when researching APIs and evaluating whether they're appropriate for a project you're working on.

Read the Docs

Before we even begin the process of integrating Ticketmaster into Balloonicorn's Flask application, we should figure out what Ticketmaster's API is capable of.

A good place to start is the Getting Started page <https://developer.ticketmaster.com/products-and-docs/apis/getting-started/>. Don't read the entire page in detail just yet, but *do* skim it — your goal is to get an idea of the various APIs that Ticketmaster has available and what each API is capable of (*hint: the information you're looking for is in a section titled,* ***Use Cases***).

According to the documentation, we should use Ticketmaster's ***Discovery API*** because it can be used to search for events by keyword in a certain location. We can use it to allow Balloonicorn's guests to search for nearby events.

Let's check out the documentation on Ticketmaster's Discovery API <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>. Read the ***Overview*** section (don't worry if you don't completely understand it) — it'll give you all the prerequisite information you need before you can start working with the API. We'll summarize the important bits below.

According to the ***Overview***, we know that

- We'll need an API key before we can start using the Discovery API
- The root URL is `https://apps.ticketmaster.com/discovery/v2/`; if we want to access a particular API ***endpoint***, we'll specify the URL by starting with the root URL and then concatenating the endpoint after the last `/`
- Ticketmaster's ***rate limit*** will only allow us to make 5000 API calls per day, at a rate of 5 requests per second. Also, they will only give us up to 1000 results.

Browse the Discovery API Endpoints

Next, let's get an idea of the specific things we can do with the Discovery API. These specific actions are provided by the API's ***endpoints***. You can trigger an action by making an HTTP request to its particular endpoint.

Endpoints are like routes in Flask — they're paths that you add to the end of an API's root URL. In the Discover API documentation <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>, you'll see a listing of all available endpoints (*hint: you'll be able to find this list after the **Overview** section; it'll be in the section that starts with **V2***). Skim the list of endpoints and find the one that allows us to search for events.

We're interested in using the ***Events Search*** (`/discovery/v2/events`) endpoint, since that's the endpoint that will allow us to "Find events and filter your search by location, date, availability, and much more".

Now we should get a better idea of how this specific endpoint works. Head over to the Event Search endpoint documentation <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/#anchor_find> and keep an eye out for the following pieces of information:

- Which HTTP method should be used to make a request to this endpoint
- The URL suffix (the route that we should add to the end of the root URL)
- Any query parameters that are permitted

With all that out of the way, we have everything we need to start making requests and seeing what sorts of results Ticketmaster gives us!

Experiment with *Events Search*

Get an API Key

In order to make requests to the ***Events Search*** endpoint, you'll need an API key.

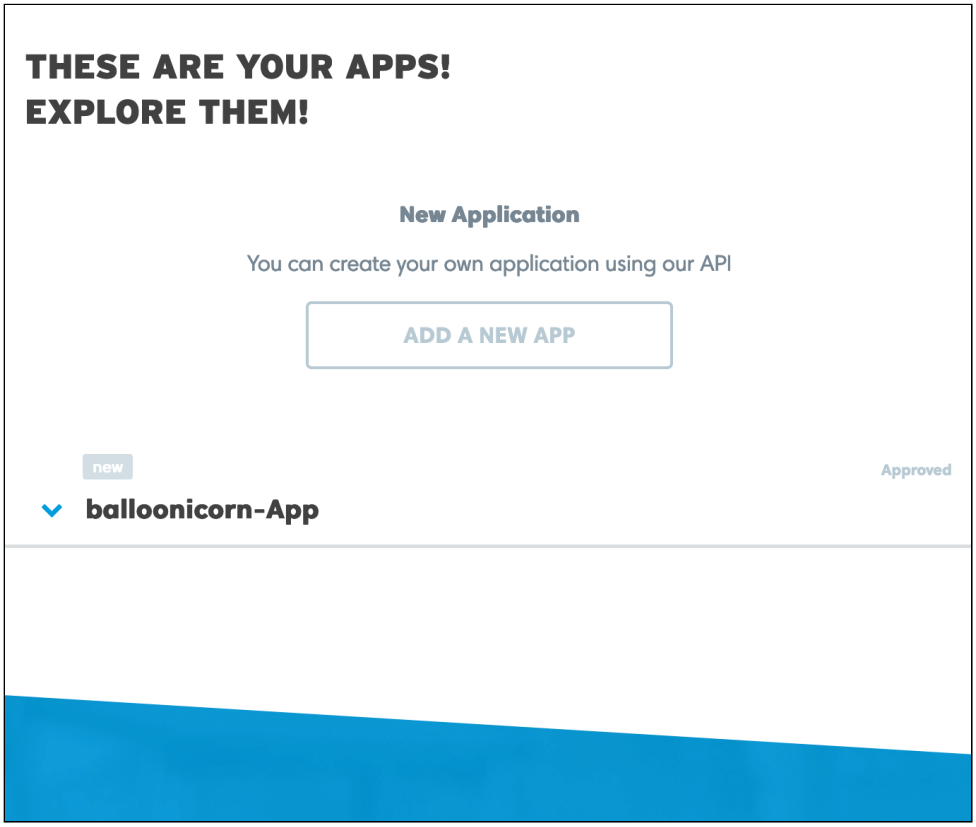
Ticketmaster's third-party developer experience is similar to that of Twitter, Reddit, and many other APIs. You need to be a registered user on Ticketmaster's developer platform before they'll give you an API key.

Go to Ticketmaster's developer login page <https://developer-acct.ticketmaster.com/user/login> to register your account. You'll be required to fill out the following info:

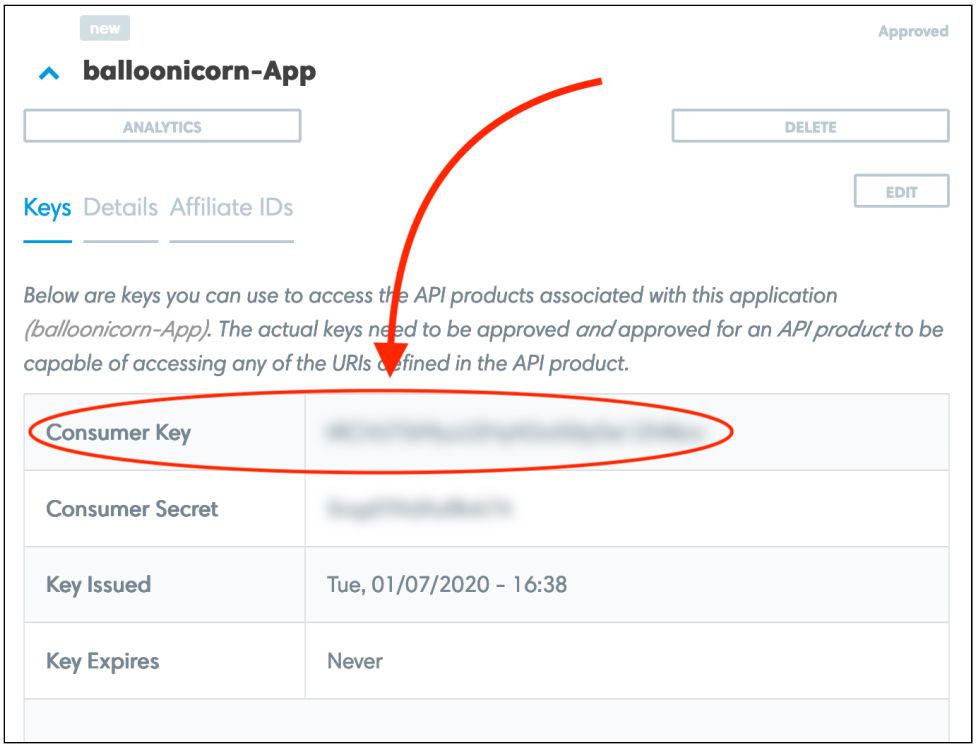
- First Name
- Last Name
- Username: choose a username that you'll be able to remember
- E-mail address
- Company Name: you can just use `Personal` as your company name (to indicate that the API key is only for personal use)
- Company Site URL: use the URL of your GitHub profile

Ticketmaster will send you a confirmation email with a link that will allow you to set your password and log in. When you get the message that your information has been saved, you've successfully set up your account and logged in.

Ticketmaster will have already generated an API key for you. To get the key, go to the My Apps page <https://developer-acct.ticketmaster.com/user/19578/apps>. Ticketmaster will list your applications below the ***Add a New App*** button. You should see one application named `yourusername-App` (see below).



Click on the blue arrow (to the left of the application name) to expand the information about your application. This is where you'll find your API key; it's the labeled under ***Consumer Key*** (we censored our key in the image below).

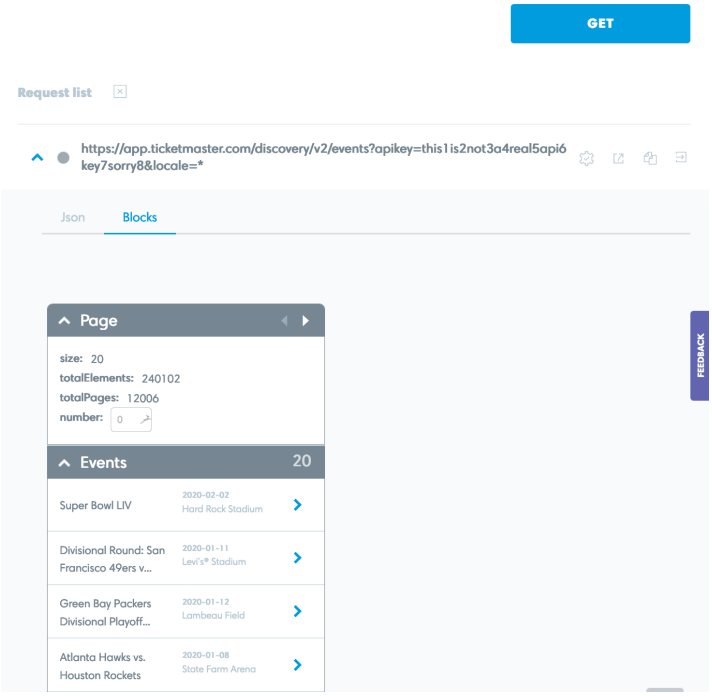


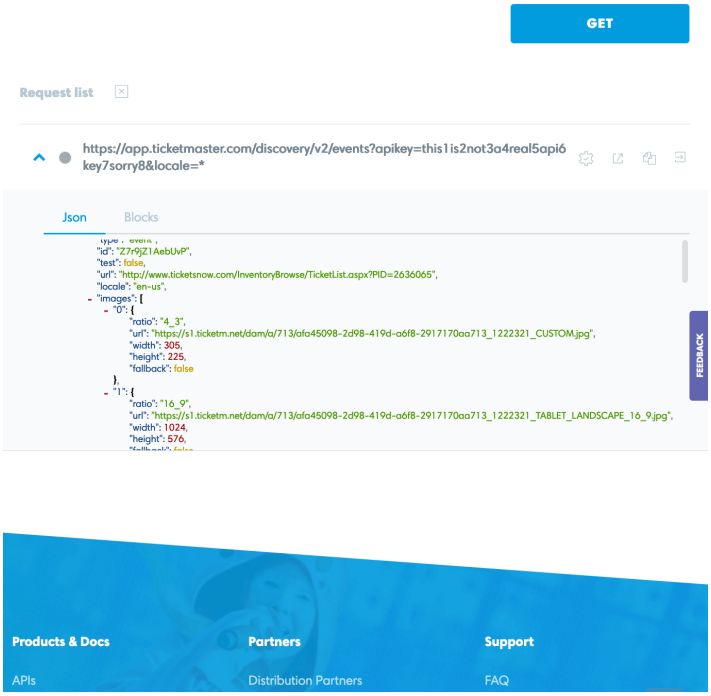
Experiment Using the API Explorer

As a convenience, most of the better-supported APIs have a browser-based API explorer. They allow you to try out different endpoints with different parameters and get an idea of how responses will look without having to write your own code.

Go to Ticketmaster's API Explorer <https://developer.ticketmaster.com/api-explorer/v2/>. The ***Event Search*** endpoint will be automatically selected for you; all you have to do is click the ***Get*** button to make a request to the endpoint.

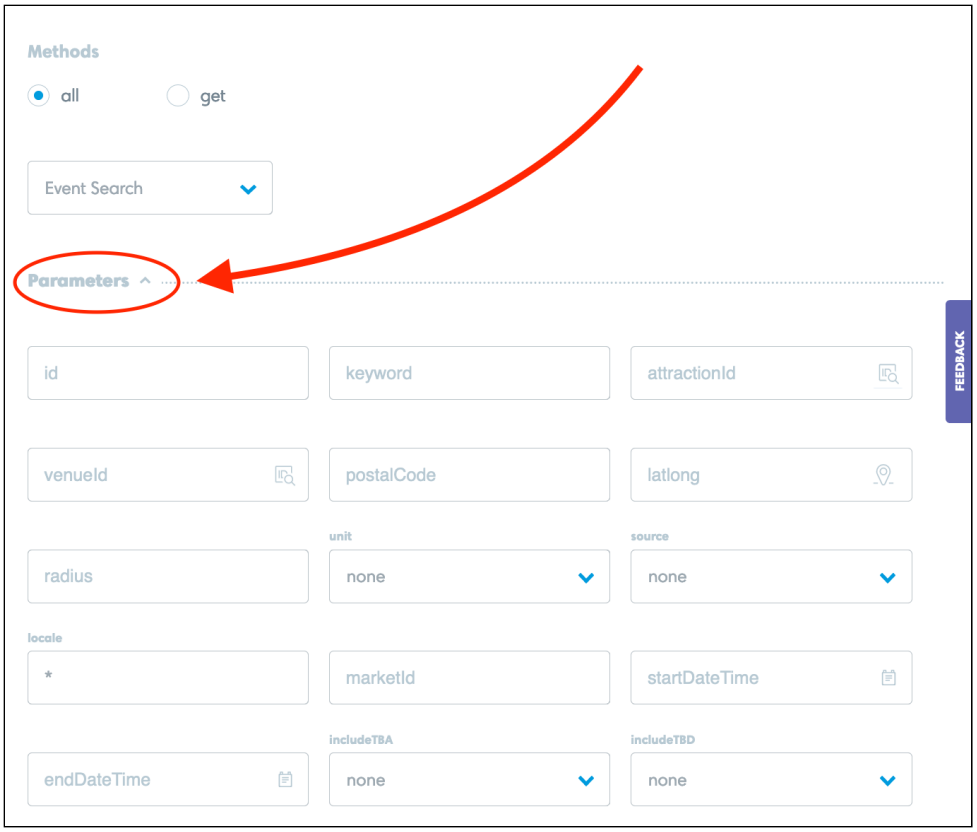
The results of your request will appear under the ***Request list*** section. It'll list the URL of the request as well as its results. You can view the results as a nicely formatted list of blocks (as seen on the left, below) or as raw JSON (as seen on the right, below).





> **Note: JSON**
>
> Notice the format of the response body. JSON, JavaScript Object Notation, is a popular data exchange format — not only for APIs, but for all sorts of data traveling around the web.

Now let's see what happens when we add parameters to the request. You can do this on the API Explorer page by expanding the ***Parameters*** section (see the image below).



Find the box labeled ***postalCode*** and type in a zip code. Then, execute the request by pressing the ***Get*** button. Again, the results of your new request should appear in the ***Request list*** section. How do these results differ from the results of your initial request?

Try this a few times with different parameters. You can even combine parameters to make your search more specific. See what results you get!

Experiment with *Events Search* in Python

Now that you're done poking around and you have a better understanding of how the ***Event Search*** endpoint works, we want to be able to access this data using Python. Lucky for us, there's a popular HTTP library for Python called Requests <https://requests.readthedocs.io/en/master/#requests-http-for-humans> that provides functions and classes for making HTTP requests in Python.

Get Familiar with *Requests*

Let's try out ***Requests*** and use it to get data from the ***Events Search*** endpoint before we integrate it with Balloonicorn's Flask app.

Make a virtual environment and activate it:

```
$ virtualenv env
$ source env/bin/activate
```

Install ***Requests*** with ***pip***:

```
(env) $ pip3 install requests
```

Whenever you install a module you're unfamiliar with, it's a good practice to try it out in the ***python3*** interpreter. With your virtual environment still active, run ***python3*** to start the interpreter:

```
(env) $ python3
```

In the Python interpreter, import ***Requests*** and save the URL of the ***Events Search*** endpoint in a variable:

```
>>> import requests
>>> url = 'https://app.ticketmaster.com/discovery/v2/events'
```

You also need to send your API key with the request as a parameter. The Requests Quickstart Guide <https://requests.readthedocs.io/en/master/user/quickstart/#passing-parameters-in-urls> shows us that we can format parameters as a dictionary. Define a dictionary called ***payload*** with the following contents (remember to replace `YOUR_API_KEY` with your actual API key):

```
>>> payload = {'apikey': 'YOUR_API_KEY'}
```

Now you're ready to make your first HTTP request with ***Requests***! According to Ticketmaster's documentation, we should use the HTTP GET method to make a request to the ***Events Search*** endpoint. According to Requests's documentation <https://requests.readthedocs.io/en/master/user/quickstart/#make-a-request>, we can make a GET request by calling a function named ***get***. It takes in the URL of the request as a string and returns a ***Response*** object. We also need to pass in the ***payload*** dictionary as a keyword argument called ***params***:

```
>>> response = requests.get(url, params=payload)
>>> response
<Response [200]>
```

If you see `<Response [200]>`, it means we have a ***Response*** object that represents a successful request. Yay!

Extract Data from the Response

The ***Response*** object has a variety of methods and attributes that can be used to access data about a response. For example, we can access the URL of the request (note that the data in ***payload*** has been correctly encoded as a query string!):

```
>>> response.url
'https://app.ticketmaster.com/discovery/v2/events?apikey=YOUR_API_KEY'
```

According to the docs <https://requests.readthedocs.io/en/master/user/quickstart/#json-response-content>, a useful method is ***Response.json***. The method can be called with no arguments; it will parse any JSON contained in the response and return it as a Python dictionary.

Call ***Response.json*** on the ***Response*** object you saved to the variable ***response***. Store its return value in a variable and access it:

```
>>> data = response.json()
>>> data
(lots of text will appear)
```

This may look like a jumbled mess at this point, but if you look closely, we can see that it's a Python dictionary!

The dictionary contains a *lot* of data — too much data! We won't use all this data in Balloonicorn's application, so let's practice extracting *useful* information from the dictionary. This can be a tedious process, so we'll do it slowly.

The whole point of the ***Events Search*** endpoint is to search for events. Let's get the list of events out of ***data*** and store it in a variable:

```
>>> events = data['_embedded']['events']
>>> events
(still lots of data, stored in a list)
```

Note that ***events*** is a list — not a dictionary. Let's get one event out of the list store it in a variable:

```
>>> event = events[0]
>>> event
(less data this time!)
```

event is a dictionary with information about one event. An easy way to see what sort of information is provided about an event is to check out the dictionary's keys:

```
>>> event.keys()
dict_keys(['name', 'type', 'id', 'test', 'url', 'locale', 'images', 'sales',
'dates', 'classifications', 'outlets', 'seatmap', '_links', '_embedded'])
```

Try indexing into ***event*** with the keys that sound interesting to you. For example, we can access the event's name like so:

```
>>> event['name']
Name of an Event
```

Add More Parameters

When you experimented with the ***Event Search*** endpoint with the API Explorer, you searched for events near a zipcode by specifying a value for the request parameter called `postalCode`. Let's do that in Python with the ***requests*** module by adding another key-value pair to our ***payload*** dictionary.

Earlier, you created a dictionary called ***payload***, but right now, it only contains your API key:

```
>>> payload
{'apikey': 'YOUR_API_KEY'}
```

Let's add the `postalCode` parameter to the dictionary. The key should be `'postalCode'` and the value should be a string that's a zipcode. For example:

```
>>> payload['postalCode'] = '94102'
>>> payload
{'apikey': 'YOUR_API_KEY', 'postalCode': '94102'}
```

Make another request with your updated ***payload***. Check out the results you recieve and notice how they differ from your initial results.

```
>>> response = request.get(url, params=payload)
>>> data = response.json()
>>> events = data['_embedded']['events']
>>> events
(a bunch of events will appear)
>>> events[0]
(data of the first event will appear)
```

Try doing this again with different parameters or combinations of parameters.

Please stop here and ask for a code review.

Integrate *Events Search* with Flask

Now back to Balloonicorn's Flask application. She'd like you to implement a feature that will allow her guests to search for potential after-party events to attend. To do so, you'll use the ***requests*** library in ***server.py*** to implement the search.

Setup

As usual, you'll need to install the project dependencies in ***requirements.txt***. Make sure your current virtual environment is still active when you do this!

Next, set up a ***git*** repo so that you can make an inital commit. Run ***git init*** to initialize the repo.

Before you make any commits, you should create your ***.gitignore*** file. Besides ignoring all the usual stuff, **make sure you add a line to ignore a file called** `secrets.sh`. This is very important! Your ***.gitignore*** should look something like this:

.gitignore

```
env
__pycache__
secrets.sh
```

This is a good time to make your first commit.

Keep Your API Key Secret

The reason you need to add ***secrets.sh*** to ***.gitignore*** is because that's where you'll store your API key. Here are the general steps you can follow to keep your secrets safe from being accidentally uploaded to GitHub.

Create a file called ***secrets.sh***. The ***.sh*** extension means that the file is a shell script — a script that contains shell commands. In other words, ***secrets.sh*** will contain code that you can run with ***bash*** or another compatible shell language (like ***zsh***).

Open ***secrets.sh*** in Sublime Text. Add the following line to the file (again, make sure you replace `YOUR_API_KEY` with your actual API key):

secrets.sh

```
export TICKETMASTER_KEY="YOUR_API_KEY"
```

The line above will initialize an environment variable called ***TICKETMASTER_KEY*** whose value will be your API key.

> **Warning: Double-quotes and single-quotes are different things in bash!**
>
> Be aware that the double-quotes (`"`) are **required** here. Don't interchange them with single-quotes! Unlike in Python where double-quotes and single-quotes can be used interchangably, in shell syntax, double-quotes and single-quotes do two different things.

Next, you'll need to

- Execute ***secrets.sh***
- Learn how to access environment variables in Python

Execute Your Shell Script

To load the ***TICKETMASTER_KEY*** variable into your shell environment, you'll need to execute ***secrets.sh***. This is similar to how you can execute Python files with ***python3***; to run shell scripts, you'll use a command called ***source***:

```
$ source secrets.sh
```

To verify that it worked, you can use ***echo*** to print the value of ***TICKETMASTER_KEY*** to the terminal:

```
$ echo $TICKETMASTER_KEY
YOUR_API_KEY
```

> **Note: When should I run `source secrets.sh`?**
>
> You only need to run `source secrets.sh` once *per terminal window*. If you forget to do this, don't worry — any Python program that requires your environment variables will just throw an error.

How to Access Environment Variables in Python

You can access environment variables form Python with the built-in ***os*** module (*reference:* Python3 Documentation: os — Miscellaneous operating system interfaces <https://docs.python.org/3/library/os.html>). Let's try using it in the Python interpreter. In your terminal, run ***python3***.

In the interpreter, import the ***os*** module:

```
>>> import os
```

Now you'll have access to a dictionary called ***os.environ***. This is where Python stores all your environment variables and their values! Check it out in the interpreter:

```
>>> os.environ
environ({'USER': 'engineering', 'PATH': 'blahblahblahblah', ...})
```

In ***os.environ***, you should be able to access the value of ***TICKETMASTER_KEY***. Try doing that it in the interpeter:

```
>>> os.environ['TICKETMASTER_KEY']
'YOUR_API_KEY'
```

Success! Now you can avoid writing your secret tokens directly in your Python code… which means you won't have to worry about accidentally uploading these secrets to GitHub. As a bonus, you now know how to acceess environment variables from Python.

> **Note: Other use-cases for environment variables**
>
> Environment variables aren't just used to keep secrets out of GitHub. They're more often used by developers to configure programs or provide a way to define user settings.
>
> For example, many programs have a *production* mode that's used to run applications that are published and a *debug* mode that's used to run applications while they're still in development. A common way to switch between either mode is to specify a value for some environment variable. For example:
>
> ```
> export NODE_ENV="development"
> # or
> export NODE_ENV="production"
> ```
>
> Also, the quotation marks aren't techinically required here. This is also valid code:
>
> ```
> export NODE_ENV=development
> # or
> export NODE_ENV=production
> ```

Implement Event Search in Flask

Run ***server.py*** and open the web application in your browser.

Navigate to the application's homepage and click on the button that says ***Click here to search for an event***. The button will take you to a route called ***/afterparty***.

At ***/afterparty***, you'll see a form that will be used to allow users to search for events on Ticketmaster. The form has fields to search by keyword, zipcode, and distance radius, but it doesn't completely work. Try clicking on the ***Look for events!*** button to submit the form. This should take you to ***/afterparty/search***. For now, it only displays test data.

Open ***server.py*** in Sublime Text. Note the two global variables — ***EVENT_SEARCH_URL*** and ***API_KEY***.

Find the ***/afterparty/search*** route. Some of the route has been implemented already. Your task is to finish implementing the logic in this route by accomplishing the task listed in the ***TODO*** comment.

Use the ***Requests*** library to make a request to the ***Event Search*** endpoint to search for events and display them to the user.

When you finish this, please ask for a code review.