Creating a Spotify playlist

SARAH JACOB

Authorization with Spotify API

Link for documentation : https://developer.spotify.com/documentation/general/guides/authorization-guide/

GET USERNAME AND CILENT ID AND SECRET:

We can get the username by simply going to the profile of the user.

https://developer.spotify.com/dashboard/applications/a7b29087b45a44b7a208b4e8fa58192e

Get the client_id and client_secret.

GET REQUEST FOR OBTAINING CODE(In browser):

https://accounts.spotify.com/authorize?client_id=(CLIENT_ID)&response_type=code&redirect_uri=http%3A%2F%2Flocalhost%3A8888%2Fcallback%2F&scope=playlist-modify-public

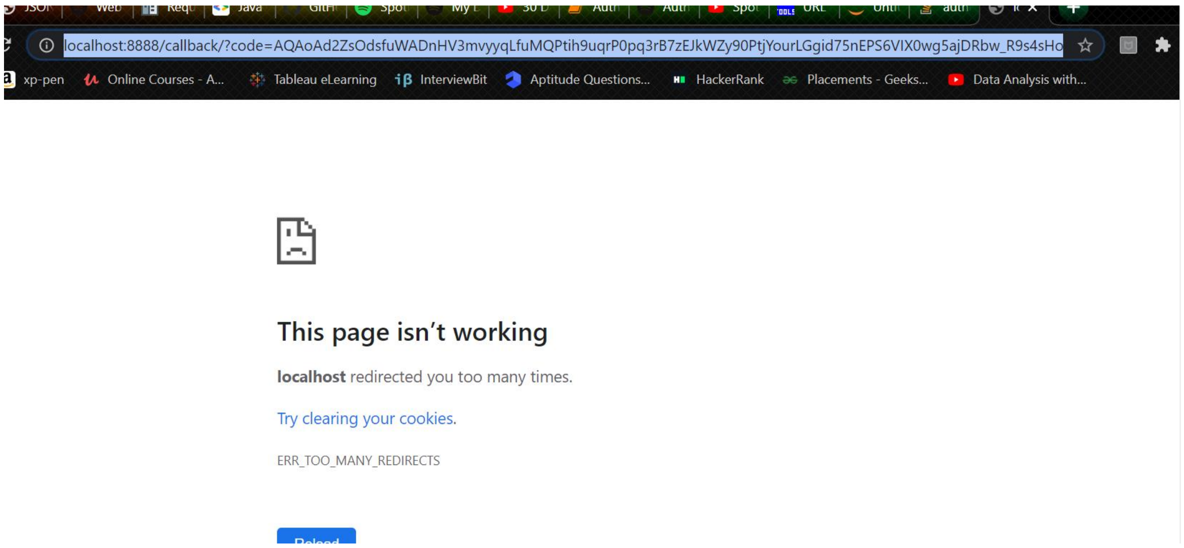
redirect_uri= https://api-university.com/	https%3A%2F%2Fapi-university.com%2F
http://localhost:8888/callback/	http%3A%2F%2Flocalhost%3A8888%2Fcallback%2F

PLEASE ENSURE THERE ARE NO SPACES BETWEEN ANY OF THE CHARACTERS!!!!

scopes= playlist-modify-public

Will return : http://localhost:8888/callback/?code=(will return a long code)

Example:



localhost:8888/callback/?code=AQAoAd2ZsOdsfuWADnHV3mvyyqLfuMQPtih9uqrP0pq3rB7zEJkWZy90PtjYourLGgid75nEPS6VIX0wg5ajDRbw_R9s4sHo

This page isn't working

localhost redirected you too many times.

Try clearing your cookies.

ERR_TOO_MANY_REDIRECTS

ENTER IN COMMAND LINE TO OBTAIN ACCESS TOKEN:

curl -H "Authorization: Basic YTdiMjkwODdiNDVhNDRiN2EyMDhiNGU4ZmE1ODE5MmU6OTVjN2I5MmFjZTE2NDk0ZDk3MTgxNTYzYWY0MTBhNWE=" -d grant_type=authorization_code -d code= (enter the code obtained) -d redirect_uri=http%3A%2F%2Flocalhost%3A8888%2Fcallback%2F https://accounts.spotify.com/api/token

Will give:

{

"access_token":" (will return an access token) ",

"token_type":"Bearer",

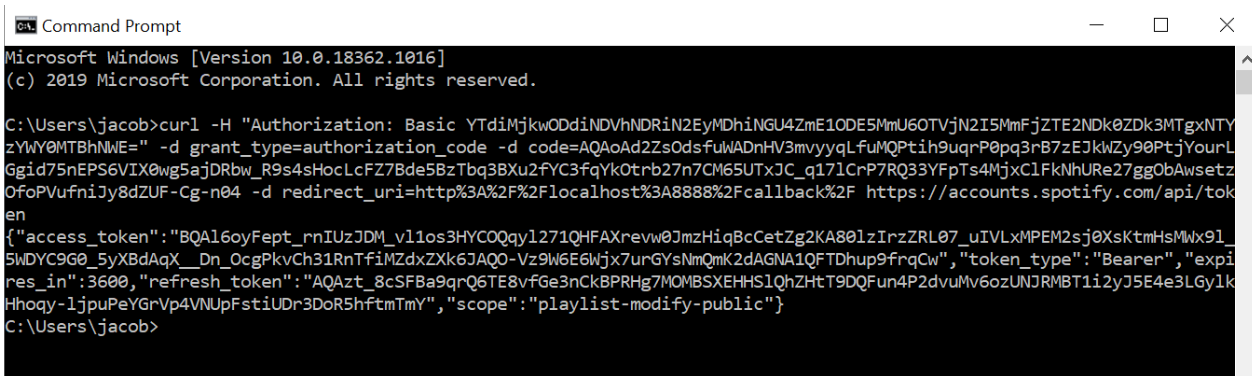
"expires_in":3600,

"refresh_token":"(refresh token)",

"scope":"playlist-modify-public"

}

Example :



```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\jacob>curl -H "Authorization: Basic YTdiMjkwODdiNDVhNDRiN2EyMDhiNGU4ZmE1ODE5MmU6OTVjN2I5MmFjZTE2NDk0ZDk3MTgxNTY
zYWY0MTBhNWE=" -d grant_type=authorization_code -d code=AQAoAd2ZsOdsfuWADnHV3mvyyqLfuMQPtih9uqrP0pq3rB7zEJkWZy90PtjYourL
Ggid75nEPS6VIX0wg5ajDRbw_R9s4sHocLcFZ7Bde5BzTbq3BXu2fYC3fqYkOtrb27n7CM65UTxJC_q17lCrP7RQ33YFpTs4MjxClFkNhURe27ggObAwsetz
OfoPVufniJy8dZUF-Cg-n04 -d redirect_uri=http%3A%2F%2Flocalhost%3A8888%2Fcallback%2F https://accounts.spotify.com/api/tok
en
{"access_token":"BQAl6oyFept_rnIUzJDM_vl1os3HYCOQqyl271QHFAXrevw0JmzHiqBcCetZg2KA80lzIrzZRL07_uIVLxMPEM2sj0XsKtmHsMWx9l_
5WDYC9G0_5yXBdAqX__Dn_OcgPkvCh31RnTfiMZdxZXk6JAQO-Vz9W6E6Wjx7urGYsNmQmK2dAGNA1QFTDhup9frqCw","token_type":"Bearer","expi
res_in":3600,"refresh_token":"AQAzt_8cSFBa9qrQ6TE8vfGe3nCkBPRHg7MOMBSXEHHSlQhZHtT9DQFun4P2dvuMv6ozUNJRMBT1i2yJ5E4e3LGylk
Hhoqy-ljpuPeYGrVp4VNUpFstiUDr3DoR5hftmTmY","scope":"playlist-modify-public"}
C:\Users\jacob>
```

TO REFRESH THE ACCESS TOKEN:

```
curl -H "Authorization: Basic
YTdiMjkwODdiNDVhNDRiN2EyMDhiNGU4ZmE1ODE5MmU6OTVjN2I5MmFjZTE2NDk0ZDk3MTgxNTYzY
WY0MTBhNWE=" -d grant_type=refresh_token -d refresh_token=(give refresh token)
https://accounts.spotify.com/api/token
```

Should return:

```
{

"access_token":"(new access token)",

"token_type":"Bearer",

"expires_in":3600,

"scope":"playlist-modify-public"

}
```

JSON- JavaScript Object Notation

Before we start, you will notice further on the excessive use of json, so here is a brief about what it is.

JSON(JavaScript Object Notation) is a lightweight data-interchange format that easy for humans to read and write. It is also easy for computers to parse and generate. JSON is based on the JavaScript programming language. It is a text format that is language independent and can be used in Python, Perl among other languages. It is primarily used to transmit data between a server and web applications**. It is primarily used to transmit data between a server and web applications. Web services and APIs use JSON format to provide public data**. JSON is built on two structures:

- A collection of name/value pairs. This is realized as an object, record, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. This is realized as an array, vector, list, or sequence.

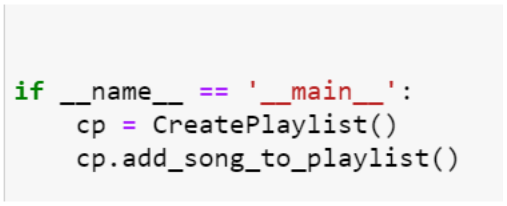
json.dump() – It convert a python dictionary into a json

json.loads() – It converts a json into a python dictionary

The ***json()*** method of the Body mixin takes a Response stream and reads it to completion. It returns a promise that resolves with the result of parsing the body text as ***JSON*** .

https://www.datacamp.com/community/tutorials/json-data-python?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=332602034358&utm_targetid=aud-299261629574:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=9061994&gclid=Cj0KCQjw7sz6BRDYARIsAPHzrNJqsrB4K4xYdBFZpHXDfnw_uvgBbhIvM6n71fvRJxDCuE5mlCiZGUMaArfgEALw_wcB

Step 1: Creating an Object of the class ‘CreatePlaylist’



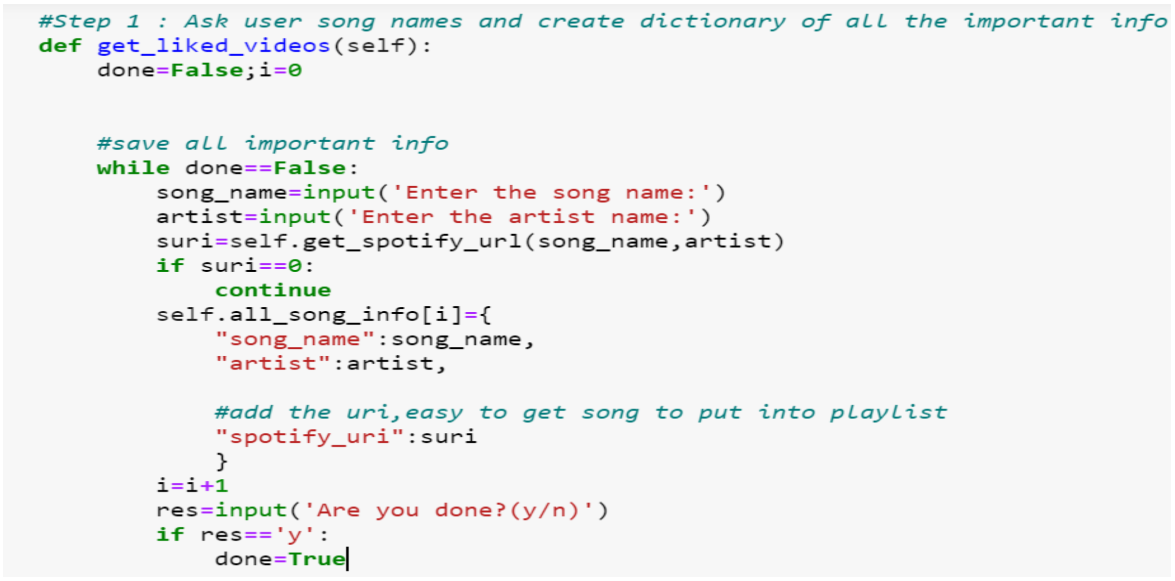
```
if __name__ == '__main__':
    cp = CreatePlaylist()
    cp.add_song_to_playlist()
```

The CreatePlaylist is the class which embodies the four functions used to create the new playlist. The functions are:

- get_liked_videos(self)
- get_spotify_url(self,song_name,artist)
- add_song_to_playlist(self)
- new_playlist(self)

We will see about these four functions in more detail later on. But the flow of the program in general is – First we call the add_song_to_playlist() function. This function further calls the get_liked_videos() function which asks the user the song and artist name of the song to be added into the new playlist. To search whether the given song is a valid song in Spotify and to retrieve its url if it is we call get_spotify_url(). In this function we use the Spotify API to search for the given song. If found the url is return else we return 0. According to the value return, the song is added into a dictionary along with their urls. Once the user is done entering the song names, the flow of the program returns to the add_song_to_playlist() function. To add these songs to a new playlist, we need to first create it and retrieve its ID in order to add songs into it. So we call the new_playlist() function which using the Spotify API creates a new playlist (we ask the user what they want to name their new playlist) and returns its ID. In the add_song_to_playlist() function we store only the urls of the songs from the initial dictionary and store it in a separate list. Then yet again using the Spotify API we give the json object of urls along with the ID pf playlist in which it adds the songs. And hence the new Playlist is created.

Step 2: Creating dictionary of songs to be added



```python
#Step 1 : Ask user song names and create dictionary of all the important info
def get_liked_videos(self):
    done=False;i=0


    #save all important info
    while done==False:
        song_name=input('Enter the song name:')
        artist=input('Enter the artist name:')
        suri=self.get_spotify_url(song_name,artist)
        if suri==0:
            continue
        self.all_song_info[i]={
            "song_name":song_name,
            "artist":artist,

            #add the uri,easy to get song to put into playlist
            "spotify_uri":suri
            }
        i=i+1
        res=input('Are you done?(y/n)')
        if res=='y':
            done=True
```

To create the dictionary of songs we use the function get_liked_videos(). Here we see an infinite loop of asking the user the song name and artist name till he/she says she done entering the song names.

When the user enters song name, we send it to get_spotify_url(). If the the function returns a url we know that the song exists and adds it to the dictionary all_song_info in which each element consists of three values – songname, artist name and url of the song. If the get_spotify_url() function returns the value '0' then we know that we could not find the song in spotify and we ignore the entry and move on to the next entry. After each song entry we ask the user whether they are done.

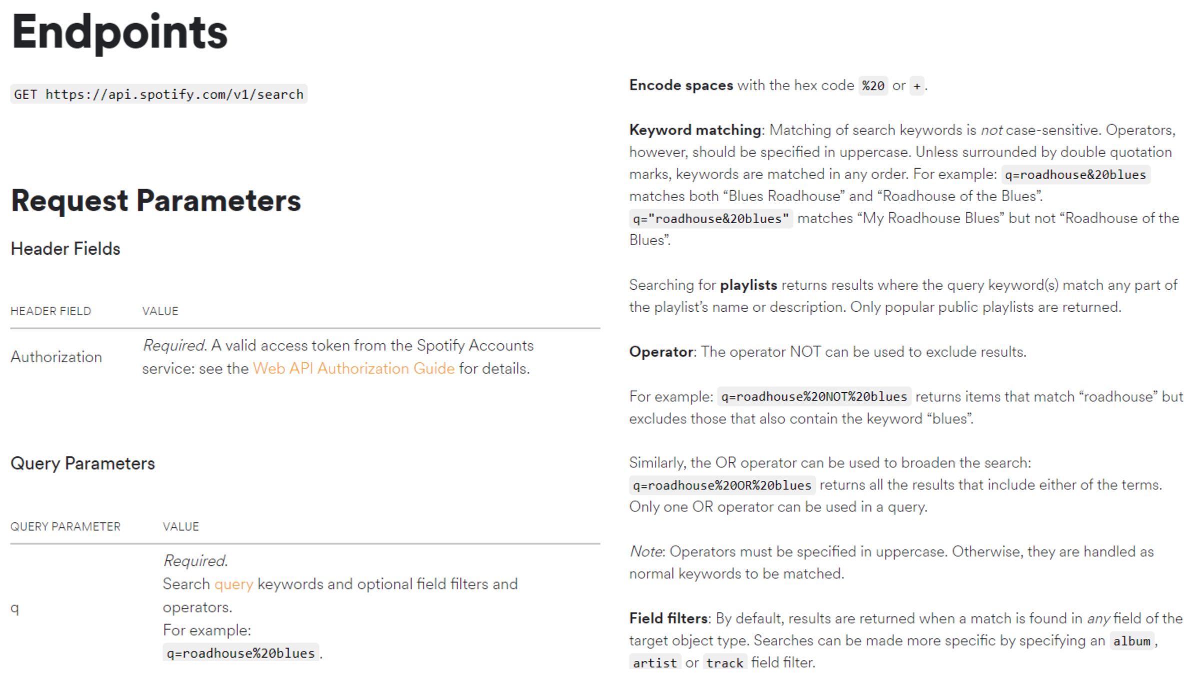
This function does to explicitly use Spotify API and simply creates a dictionary with the values given to it.

Step 3: Searching a song in Spotify

To search for a song in Spotify, we use the function get_spotify_url(). To search for a song in Spotify we access the Spotify API using the GET request using the following details mentioned in the Spotify Documentation.

https://developer.spotify.com/documentation/web-api/reference/search/search/

To search a song we use the endpoint:



Endpoints

`GET https://api.spotify.com/v1/search`

Request Parameters

Header Fields

HEADER FIELD	VALUE
Authorization	*Required.* A valid access token from the Spotify Accounts service: see the Web API Authorization Guide for details.

Query Parameters

QUERY PARAMETER	VALUE
q	*Required.* Search query keywords and optional field filters and operators. For example: `q=roadhouse%20blues`.

Encode spaces with the hex code `%20` or `+`.

Keyword matching: Matching of search keywords is *not* case-sensitive. Operators, however, should be specified in uppercase. Unless surrounded by double quotation marks, keywords are matched in any order. For example: `q=roadhouse&20blues` matches both "Blues Roadhouse" and "Roadhouse of the Blues". `q="roadhouse&20blues"` matches "My Roadhouse Blues" but not "Roadhouse of the Blues".

Searching for **playlists** returns results where the query keyword(s) match any part of the playlist's name or description. Only popular public playlists are returned.

Operator: The operator NOT can be used to exclude results.

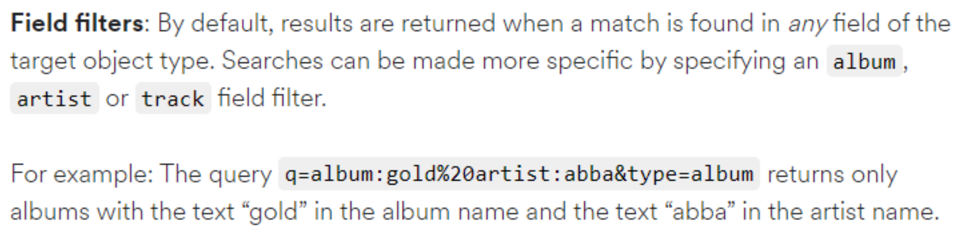
For example: `q=roadhouse%20NOT%20blues` returns items that match "roadhouse" but excludes those that also contain the keyword "blues".

Similarly, the OR operator can be used to broaden the search: `q=roadhouse%20OR%20blues` returns all the results that include either of the terms. Only one OR operator can be used in a query.

Note: Operators must be specified in uppercase. Otherwise, they are handled as normal keywords to be matched.

Field filters: By default, results are returned when a match is found in *any* field of the target object type. Searches can be made more specific by specifying an `album`, `artist` or `track` field filter.

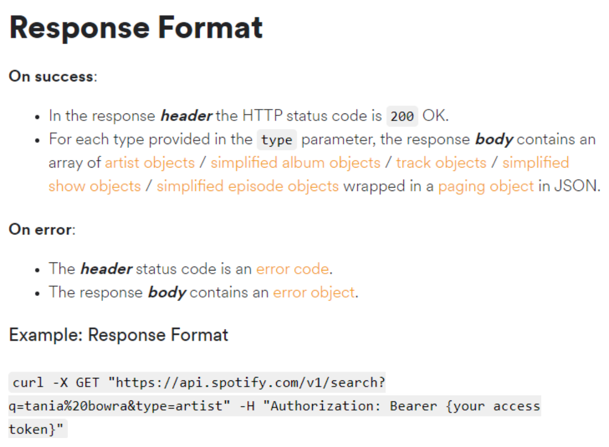
Under q (query), we have the filter options for artist,album and track



Field filters: By default, results are returned when a match is found in *any* field of the target object type. Searches can be made more specific by specifying an `album`, `artist` or `track` field filter.

For example: The query `q=album:gold%20artist:abba&type=album` returns only albums with the text "gold" in the album name and the text "abba" in the artist name.

Note : That when we encode this into url the ':' will be converted into '%3A'.



Response Format

On success:

- In the response ***header*** the HTTP status code is `200` OK.
- For each type provided in the `type` parameter, the response ***body*** contains an array of artist objects / simplified album objects / track objects / simplified show objects / simplified episode objects wrapped in a paging object in JSON.

On error:

- The ***header*** status code is an error code.
- The response ***body*** contains an error object.

Example: Response Format

```
curl -X GET "https://api.spotify.com/v1/search?
q=tania%20bowra&type=artist" -H "Authorization: Bearer {your access
token}"
```

The response will return a list of all the songs that meet the requirments. But we want to add only the first song in this list.

Following the documentation, we use it in the function as follows:

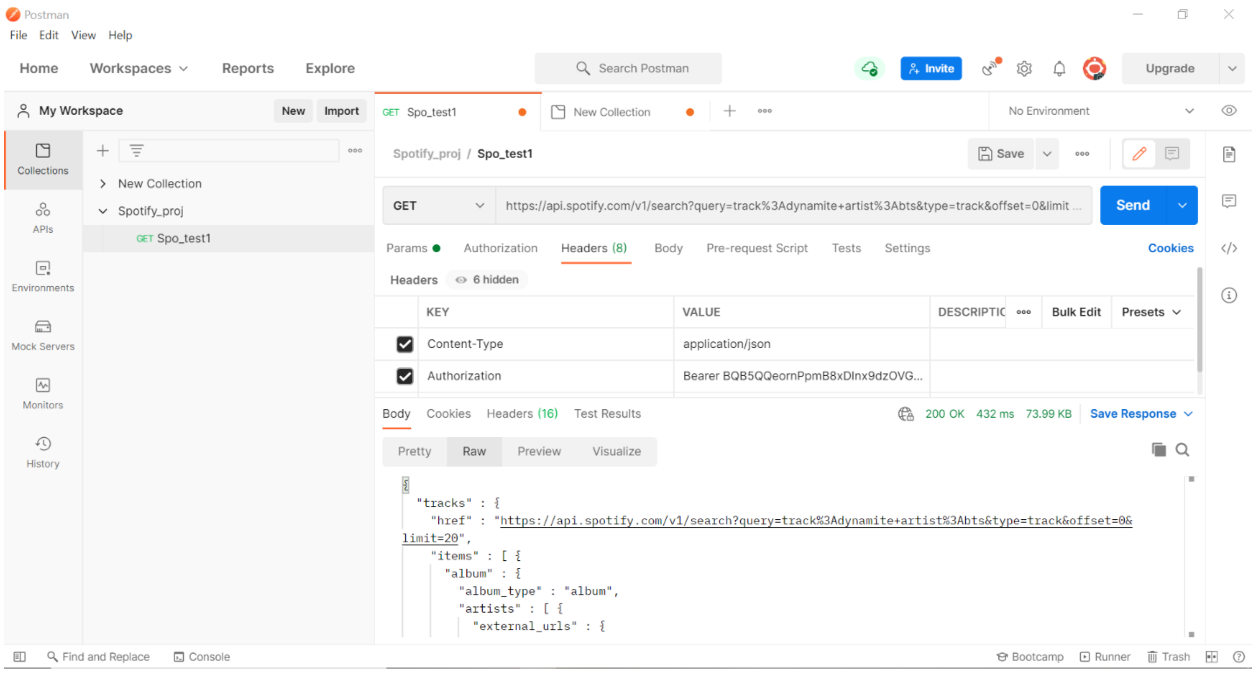
```
#Step 4 : Search For the song
def get_spotify_url(self,song_name,artist):
    query="https://api.spotify.com/v1/search?query=track%3A{}+artist%3A{}&type=track&offset=0&limit=20".format(
        song_name,artist)
    response=requests.get(query,
                          headers={
                                "Content-Type":"application/json",
                                "Authorization":"Bearer {}".format(self.s_token)

                                })

    response_json=response.json()
    #print(response_json)
    song=response_json['tracks']['items']
    #print(song)
    try:
        url=song[0]['uri']
        return url
    except:
        print('Song doesnt exist or spelt wrong...Try again')
        return 0
```

We create the GET request using the get() in request in-built module . We read and parse the data using `json()`, then read values out of the resulting objects as you'd expect and insert them into list items to display our product data. We create a dictionary of only the ‘track’ and ‘items’ details out of all the data received from the json object. Out the list of songs we choose the first in the list and return its uri value. Now if Spotify was unable to find a match for the song, then it returns ‘None’ which would pass an Exception. In the function, we catch this exception and handle it in the ‘try except block’. If song is not found, we ask the user to re-enter the song name since they might have mis-spelt it.

USING POSTMAN



Spotify_proj / Spo_test1

GET https://api.spotify.com/v1/search?query=track%3Adynamite+artist%3Abts&type=track&offset=0&limit ...

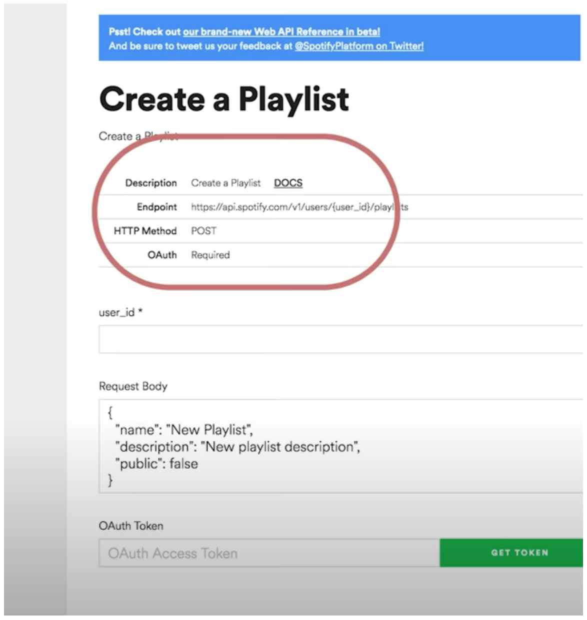
	KEY	VALUE	DESCRIPTIC
✓	Content-Type	application/json	
✓	Authorization	Bearer BQB5QQeornPpmB8xDInx9dzOVG...	

200 OK 432 ms 73.99 KB

```
{
  "tracks" : {
    "href" : "https://api.spotify.com/v1/search?query=track%3Adynamite+artist%3Abts&type=track&offset=0&
limit=20",
    "items" : [ {
      "album" : {
        "album_type" : "album",
        "artists" : [ {
          "external_urls" : {
```

Step 4 : Creating a playlist in Spotify

https://developer.spotify.com/documentation/web-api/reference/playlists/create-playlist/



Psst! Check out our brand-new Web API Reference in beta!
And be sure to tweet us your feedback at @SpotifyPlatform on Twitter!

Create a Playlist

Create a Playlist

Description	Create a Playlist DOCS
Endpoint	https://api.spotify.com/v1/users/{user_id}/playlists
HTTP Method	POST
OAuth	Required

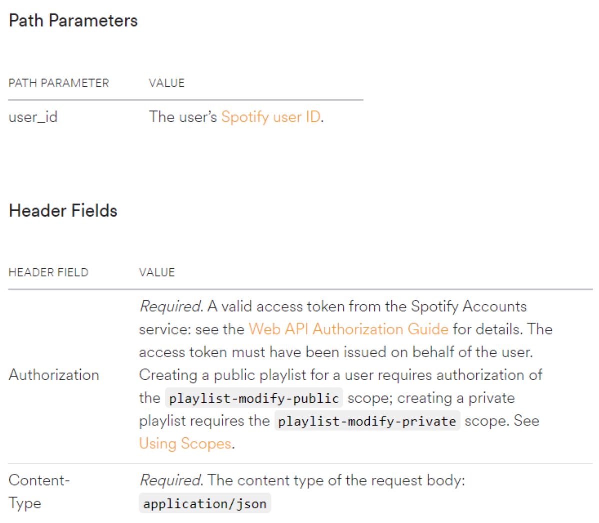
user_id *

Request Body

```
{
 "name": "New Playlist",
 "description": "New playlist description",
 "public": false
}
```

OAuth Token

OAuth Access Token



Path Parameters

PATH PARAMETER	VALUE
user_id	The user's Spotify user ID.

Header Fields

HEADER FIELD	VALUE
Authorization	*Required.* A valid access token from the Spotify Accounts service: see the Web API Authorization Guide for details. The access token must have been issued on behalf of the user. Creating a public playlist for a user requires authorization of the `playlist-modify-public` scope; creating a private playlist requires the `playlist-modify-private` scope. See Using Scopes.
Content-Type	*Required.* The content type of the request body: `application/json`

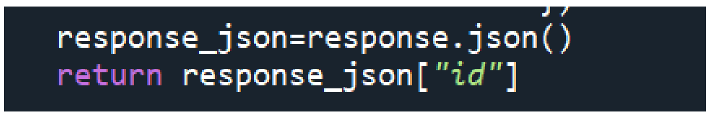
We create a post request with the endpoint given in the description along with the parameters necessary in the data and header (You can see that in the link). We ask the user what they would like to name their playlist and pass the value along with the rest of the request body data.

```python
#Step 3 : Create A New Playlist
def new_playlist(self):
    name=input('What would you like to name your playlist:')
    request_body = json.dumps({
        "name": name,
        "description": "All Liked Youtube Videos",
        "public": True
    })

    query = "https://api.spotify.com/v1/users/{}/playlists".format(
        self.user_id)
    response = requests.post(
        query,
        data=request_body,
        headers={
            "Content-Type": "application/json",
            "Authorization": "Bearer {}".format(self.s_token)
        }
    )
    response_json = response.json()

    # playlist id
    return response_json["id"]
```

When we execute the response request, it returns details about the playlist created. We need to save the id of the playlist created in order to perform actions on that playlist.

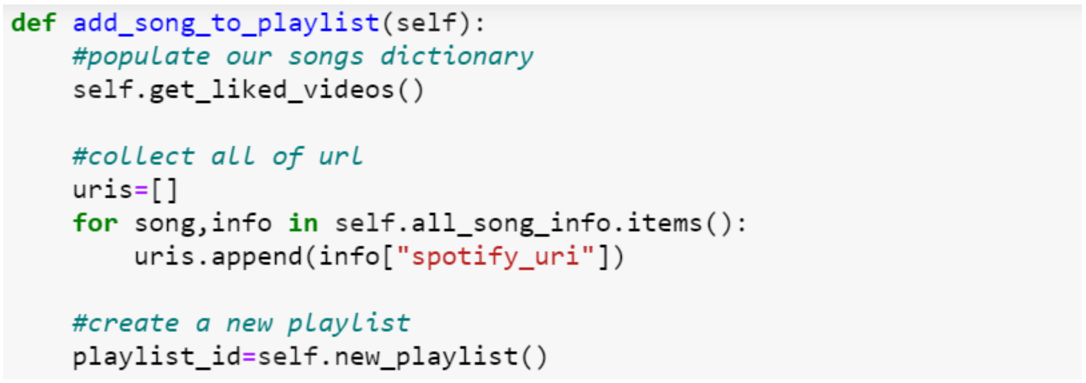


```
    response_json=response.json()
    return response_json["id"]
```

Step 5 : Adding the songs into the new playlist

https://developer.spotify.com/documentation/web-api/reference/playlists/add-tracks-to-playlist/

To perform this the add_song_to_playlist() function first calls the two functions - get_liked_videos() and new_playlist() to obtain the dictionary of songs and the ID of the playlist into which we are going to add the songs. We extract all the urls of the songs from the dictionary and store it in a separate list.

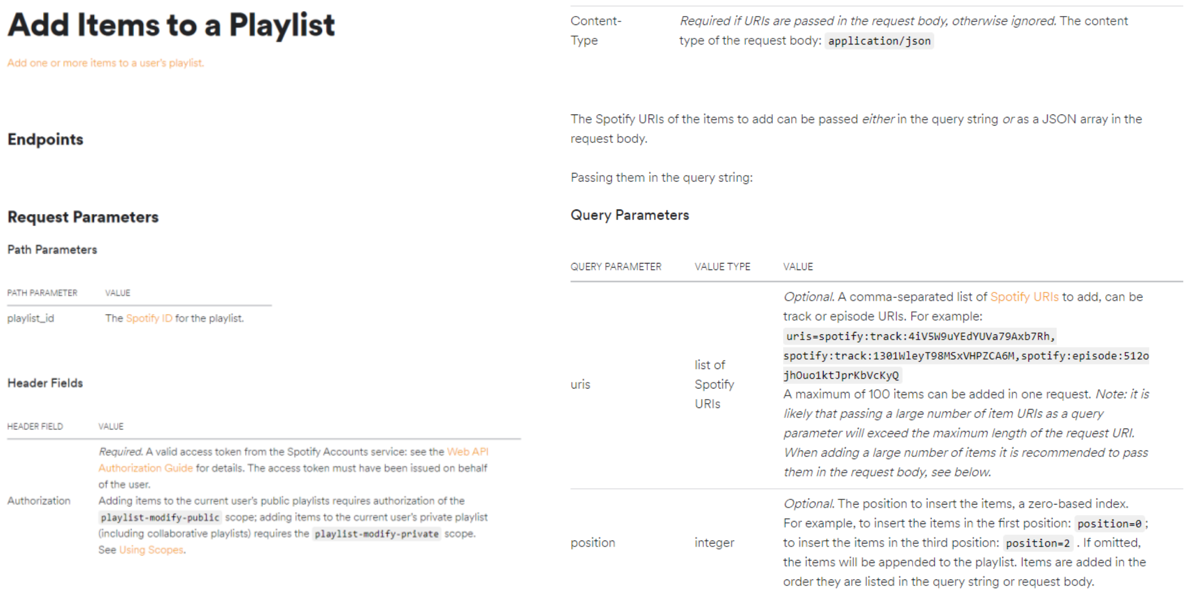


```python
def add_song_to_playlist(self):
    #populate our songs dictionary
    self.get_liked_videos()

    #collect all of url
    uris=[]
    for song,info in self.all_song_info.items():
        uris.append(info["spotify_uri"])

    #create a new playlist
    playlist_id=self.new_playlist()
```

After this we create a POST request to access the Spotify API to add the specified song urls into the playlist.



Add Items to a Playlist

Add one or more items to a user's playlist.

Endpoints

Request Parameters

Path Parameters

PATH PARAMETER	VALUE
playlist_id	The Spotify ID for the playlist.

Header Fields

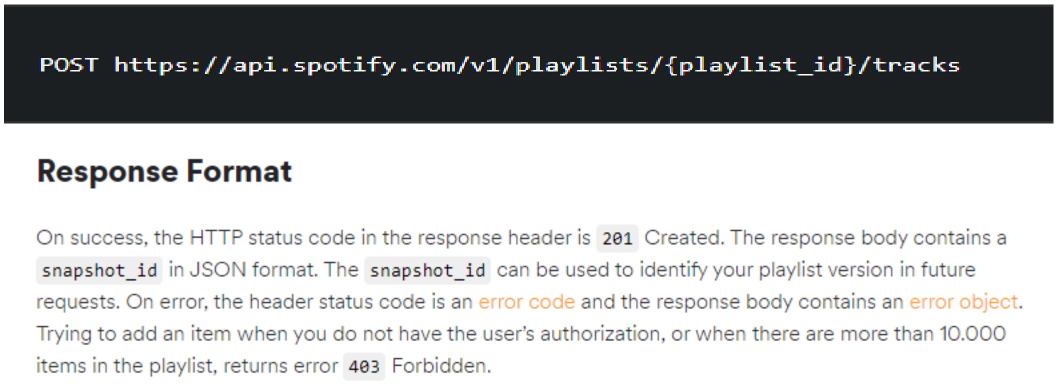
HEADER FIELD	VALUE
Authorization	*Required.* A valid access token from the Spotify Accounts service: see the Web API Authorization Guide for details. The access token must have been issued on behalf of the user. Adding items to the current user's public playlists requires authorization of the `playlist-modify-public` scope; adding items to the current user's private playlist (including collaborative playlists) requires the `playlist-modify-private` scope. See Using Scopes.
Content-Type	*Required if URIs are passed in the request body, otherwise ignored.* The content type of the request body: `application/json`

The Spotify URIs of the items to add can be passed *either* in the query string *or* as a JSON array in the request body.

Passing them in the query string:

Query Parameters

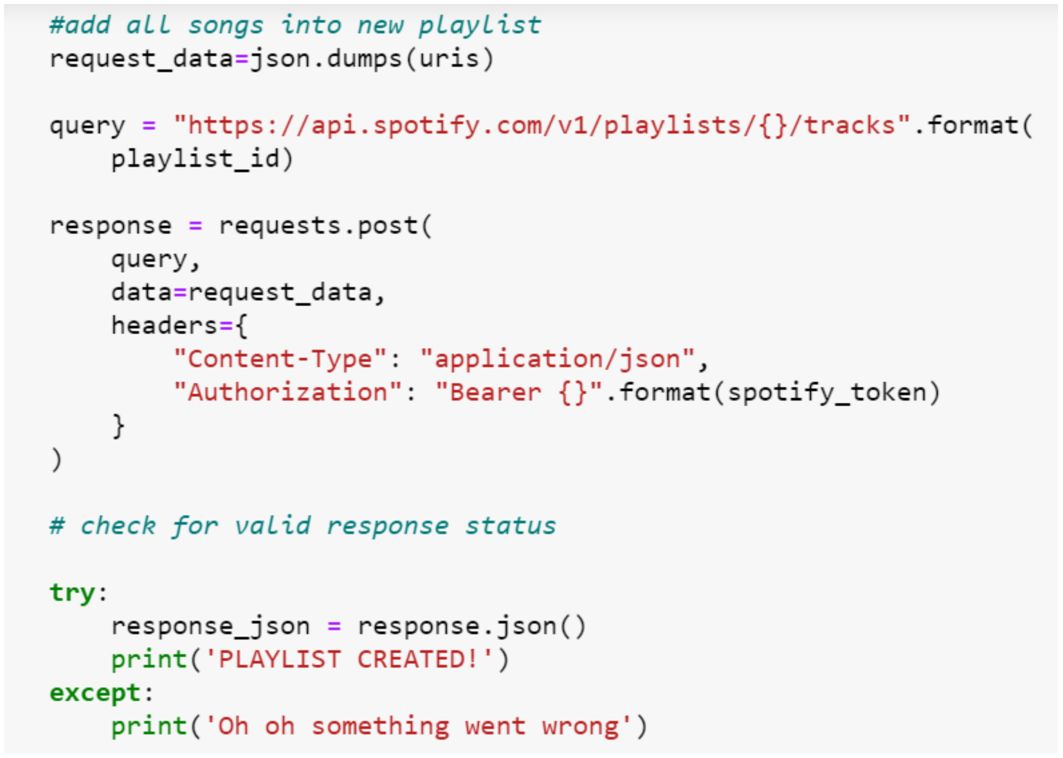
QUERY PARAMETER	VALUE TYPE	VALUE
uris	list of Spotify URIs	*Optional.* A comma-separated list of Spotify URIs to add, can be track or episode URIs. For example: `uris=spotify:track:4iV5W9uYEdYUVa79Axb7Rh, spotify:track:1301WleyT98MSxVHPZCA6M,spotify:episode:512ojhOuo1ktJprKbVcKyQ` A maximum of 100 items can be added in one request. *Note: it is likely that passing a large number of item URIs as a query parameter will exceed the maximum length of the request URI. When adding a large number of items it is recommended to pass them in the request body, see below.*
position	integer	*Optional.* The position to insert the items, a zero-based index. For example, to insert the items in the first position: `position=0`; to insert the items in the third position: `position=2` . If omitted, the items will be appended to the playlist. Items are added in the order they are listed in the query string or request body.



```
POST https://api.spotify.com/v1/playlists/{playlist_id}/tracks
```

Response Format

On success, the HTTP status code in the response header is `201` Created. The response body contains a `snapshot_id` in JSON format. The `snapshot_id` can be used to identify your playlist version in future requests. On error, the header status code is an error code and the response body contains an error object. Trying to add an item when you do not have the user's authorization, or when there are more than 10.000 items in the playlist, returns error `403` Forbidden.



```python
#add all songs into new playlist
request_data=json.dumps(uris)

query = "https://api.spotify.com/v1/playlists/{}/tracks".format(
    playlist_id)

response = requests.post(
    query,
    data=request_data,
    headers={
        "Content-Type": "application/json",
        "Authorization": "Bearer {}".format(spotify_token)
    }
)

# check for valid response status

try:
    response_json = response.json()
    print('PLAYLIST CREATED!')
except:
    print('Oh oh something went wrong')
```