Batch Data Processing from an API

In this lab, you will learn how to interact with the Spotify API and extract data from the API in a batch way. You will explore what pagination means and how to send an API request that requires authorization.

1 - Create a Spotify APP

To get access to the API resources, you need to create a Spotify account if you don't already have one. A trial account will be enough to complete this lab.

1. Go to https://developer.spotify.com/, create an account and log in.
2. Click on the account name in the right-top corner and then click on **Dashboard**.
3. Create a new APP using the following details:
    - App name: `dec2w2a1-spotify-app`
    - App description: `spotify app to test the API`
    - Website: leave empty
    - Redirect URIs: `http://localhost:3000`
    - API to use: select `Web API`
4. Click on **Save** button. If you get an error message saying that your account is not ready, you can log out, wait for a few minutes and then repeat again steps 2-4.
5. In the App Home page click on **Settings** and reveal `Client ID` and `Client secret` . Store them in the `src/env` file provided in this lab. Make sure to save the `src/env` file using `Ctrl + S` or `Cmd + S` .

Here's the link to the Spotify API documentation that you can refer to while you're working on the lab's exercises. The required information to complete the tasks will be given during the lab. You will interact with two resources:

- New album releases in the first and second parts (endpoint);
- Album tracks in the second part (endpoint).

Table of Contents

2 - Understand the Basics of APIs

Several packages in Python allow you to request data from an API; in this lab, you will use the `requests` package, which is a popular and versatile library to perform HTTP requests. It provides a simple and easy-to-use way to interact with web services and APIs. Let's load the required packages:

In [1]:
```
import os
from typing import Dict, Any, Callable

from dotenv import load_dotenv
import json
import requests
```

2.1 - Get Token

The first step when working with an API is to understand the authentication process. For that, the Spotify APP generates a Client ID and a Client secret that you will use to generate an access token. The access token is a string that contains the credentials and permissions that you can use to access a given resource. You can find more about it in the API documentation. Since each API is developed with a particular purpose, it is necessary for you to always read and understand the nuances of each API so you can access the data responsibly. Throughout this lab, you will be provided with several links to the documentation and you are encouraged to read them. (During the lab session, you may quickly skim through the links, but you can always check them in more details after the lab session).

Let's create some variables to hold the values of the client_id and client_secret that you stored in the src/env file.

In [3]:
```
load_dotenv('./src/env', override=True)

CLIENT_ID = os.getenv('CLIENT_ID')
CLIENT_SECRET = os.getenv('CLIENT_SECRET')
```

The `get_token` function below takes a Client ID, Client secret and a URL as input, and performs a POST request to that URL to obtain an access token using the client credentials. Run the following cell to get the access token.

In [4]:
```
def get_token(client_id: str, client_secret: str, url: str) -> Dict[Any, Any]:
    """Allows to perform a POST request to obtain an access token

    Args:
        client_id (str): App client id
        client_secret (str): App client secret
        url (str): URL to perform the post request

    Returns:
        Dict[Any, Any]: Dictionary containing the access token
    """

    headers = {
        "Content-Type": "application/x-www-form-urlencoded"
    }

    payload = {
                "grant_type": "client_credentials",
                "client_id": client_id,
                "client_secret": client_secret
               }

    try:
        response = requests.post(url=url, headers=headers, data=payload)
        print(type(response))
        response.raise_for_status()
        response_json = json.loads(response.content)

        return response_json

    except Exception as err:
        print(f"Error: {err}")
        return {}

URL_TOKEN="https://accounts.spotify.com/api/token"
token = get_token(client_id=CLIENT_ID, client_secret=CLIENT_SECRET, url=URL_TOKE

print(token)
```

```
<class 'requests.models.Response'>
{'access_token': 'BQBIs0HXgJHwnGHGFYnhAGRJ3ukJEgCoy5uV__2JmwNtxb0uJ-_chsf_7_IT7xV
_AFh4gn9k98t-X2W4XFq0EPgAZGx0v9ttxPEPrVHH_K3EHSp0-YOQodPR90WBNqMmELkqXLhbewY', 't
oken_type': 'Bearer', 'expires_in': 3600}
```

You can see that you are provided with a temporary access token. The `expires_in` field tells you the duration of this token in seconds. When this token expires, your requests will fail and an error object will be returned to you holding a status code of 401. This status code means that the request is unauthorized.

Whenever you send an API request to the spotify API, you need to include in the request the access token, as an authorization header following a certain format. You are provided with the function `get_auth_header`. This function expects the access token and returns the authorization header that can be included in the API request.

Make sure to run the following cell to declare the function `get_auth_header`, which you will use throughout this lab.

In [5]:
```
def get_auth_header(access_token: str) -> Dict[str, str]:
    return {"Authorization": f"Bearer {access_token}"}
```

Now, let's use the token to perform a request to access the first resource, which is the new releases.

2.2 - Get New Releases

Exercise 1

Follow the instructions to complete the `get_new_releases` function:

1. Call the function `get_auth_header` and pass to it the access token (which is specified as input to the `get_new_releases` function). Save the output of `get_auth_header` to a variable called `headers`.
2. You are provided with the URL in the `request_url` variable. Use this URL and the header from the previous step to perform a `get()` request.
3. Request `response` is an object of type `requests.models.Response`. This object has a method named `json()` that allows you to transform the response content into a JSON object or plain Python dictionary. Use this method on the `response` object to return the content as a Python dictionary.

Then you will use the provided `URL_NEW_RELEASES` URL or endpoint to perform calls to the API, passing the `access_token` value from the `token` object that you obtained before.

In [6]:
```
def get_new_releases(url: str, access_token: str, offset: int=0, limit: int=20,
    """Perform get() request to new releases endpoint

    Args:
        url (str): Base url for the request
        access_token (str): Access token
        offset (int, optional): Page offset for pagination. Defaults to 0.
        limit (int, optional): Number of elements per page. Defaults to 20.
        next (str, optional): Next URL to perform next request. Defaults to "".

    Returns:
        Dict[Any, Any]: Request response
    """

    if next == "":
        request_url = f"{url}?offset={offset}&limit={limit}"
    else:
        request_url = f"{next}"

    ### START CODE HERE ### (~ 4 lines of code)
    # Call get_auth_header() function and pass the access token.
    headers = get_auth_header (access_token=access_token)

    try:
        # Perform a get() request using the request_url and headers.
        response = requests.get(url=request_url, headers=headers)
```

```
        # Use json() method over the response to return it as Python dictionary.
        return response.json()
    ### END CODE HERE ###

    except Exception as err:
        print(f"Error requesting data: {err}")
        return {'error': err}

URL_NEW_RELEASES = "https://api.spotify.com/v1/browse/new-releases"

# Note: the `access_token` value from the dictionary `token` can be retrieved ei
releases_response = get_new_releases(url=URL_NEW_RELEASES, access_token=token.ge
```

The result you get is a JSON object that was transformed into a python dictionary. You can explore the structure of the response you get:

In [7]:
```
releases_response.keys()
```

Out[7]:
```
dict_keys(['albums'])
```

In [8]:
```
releases_response.get('albums').keys()
```

Out[8]:
```
dict_keys(['href', 'items', 'limit', 'next', 'offset', 'previous', 'total'])
```

Each API manages responses in its own way so it is highly recommended to read the documentation and understand the nuances behind the API endpoints you are working with. In this case, you see some fields such as `'href'` under the `'albums'` field, which tells you the URL used for the request you just sent.

In [9]:
```
releases_response.get('albums').get('total')
```

Out[9]:
```
100
```

You can see that there are two parameters: `offset` and `limit` that were added to the endpoint. Those parameters are the base of pagination in this API endpoint. We will take a look at them later.

You can also explore the returned items using the `'items'` field under `'albums'`. This will return a list of items, you can take a look at the number of items returned:

In [10]:
```
len(releases_response.get('albums').get('items'))
```

Out[10]:
```
20
```

Explore the items:

In [11]:
```
releases_response.get('albums').get('items')[0]
```

Out[11]:

```
{'album_type': 'album',
 'artists': [{'external_urls': {'spotify': 'https://open.spotify.com/artist/06H
L4z0CvFAxyc27GXpf02'},
   'href': 'https://api.spotify.com/v1/artists/06HL4z0CvFAxyc27GXpf02',
   'id': '06HL4z0CvFAxyc27GXpf02',
   'name': 'Taylor Swift',
   'type': 'artist',
   'uri': 'spotify:artist:06HL4z0CvFAxyc27GXpf02'}],
 'available_markets': ['AR',
  'AU',
  'AT',
  'BE',
  'BO',
  'BR',
  'BG',
  'CA',
  'CL',
  'CO',
  'CR',
  'CY',
  'CZ',
  'DK',
  'DO',
  'DE',
  'EC',
  'EE',
  'SV',
  'FI',
  'FR',
  'GR',
  'GT',
  'HN',
  'HK',
  'HU',
  'IS',
  'IE',
  'IT',
  'LV',
  'LT',
  'LU',
  'MY',
  'MT',
  'MX',
  'NL',
  'NZ',
  'NI',
  'NO',
  'PA',
  'PY',
  'PE',
  'PH',
  'PL',
  'PT',
  'SG',
  'SK',
  'ES',
  'SE',
  'CH',
  'TW',
  'TR',
```

```
'UY',
'US',
'GB',
'AD',
'LI',
'MC',
'ID',
'JP',
'TH',
'VN',
'RO',
'IL',
'ZA',
'SA',
'AE',
'BH',
'QA',
'OM',
'KW',
'EG',
'MA',
'DZ',
'TN',
'LB',
'JO',
'PS',
'IN',
'KZ',
'MD',
'UA',
'AL',
'BA',
'HR',
'ME',
'MK',
'RS',
'SI',
'KR',
'BD',
'PK',
'LK',
'GH',
'KE',
'NG',
'TZ',
'UG',
'AG',
'AM',
'BS',
'BB',
'BZ',
'BT',
'BW',
'BF',
'CV',
'CW',
'DM',
'FJ',
'GM',
'GE',
```

```
'GD',
'GW',
'GY',
'HT',
'JM',
'KI',
'LS',
'LR',
'MW',
'MV',
'ML',
'MH',
'FM',
'NA',
'NR',
'NE',
'PW',
'PG',
'WS',
'SM',
'ST',
'SN',
'SC',
'SL',
'SB',
'KN',
'LC',
'VC',
'SR',
'TL',
'TO',
'TT',
'TV',
'VU',
'AZ',
'BN',
'BI',
'KH',
'CM',
'TD',
'KM',
'GQ',
'SZ',
'GA',
'GN',
'KG',
'LA',
'MO',
'MR',
'MN',
'NP',
'RW',
'TG',
'UZ',
'ZW',
'BJ',
'MG',
'MU',
'MZ',
'AO',
```

```
  'CI',
  'DJ',
  'ZM',
  'CD',
  'CG',
  'IQ',
  'LY',
  'TJ',
  'VE',
  'ET',
  'XK'],
 'external_urls': {'spotify': 'https://open.spotify.com/album/1Mo4aZ8pdj6L1jx8z
SwJnt'},
 'href': 'https://api.spotify.com/v1/albums/1Mo4aZ8pdj6L1jx8zSwJnt',
 'id': '1Mo4aZ8pdj6L1jx8zSwJnt',
 'images': [{'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e025076e4160d018e378f488c33',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d000048515076e4160d018e378f488c33',
   'width': 64},
  {'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b2735076e4160d018e378f488c33',
   'width': 640}],
 'name': 'THE TORTURED POETS DEPARTMENT',
 'release_date': '2024-04-18',
 'release_date_precision': 'day',
 'total_tracks': 16,
 'type': 'album',
 'uri': 'spotify:album:1Mo4aZ8pdj6L1jx8zSwJnt'}
```

2.3 - Pagination

If you print `releases_response`, you can see the following fields:

```
{
...,
'limit': 20,
'next': 'https://api.spotify.com/v1/browse/new-releases?
offset=20&limit=20',
'offset': 0,
'previous': None,
'total': 100
}
```

Although there is a total of 100 available items to be returned, only 20 were returned. This is established by the `limit` parameter and those were the 20 items you just counted before. This limit on the number of elements returned is a common feature of several APIs and although in some cases you can modify such a limit, a good practice is to use it with **pagination** to get all the elements that can be returned.

Each API handles pagination differently. For Spotify, the requests response provides you with two fields that allow you to query the different pages of your request: `previous` and `next`. These two fields will return the URL to the previous or next page respectively and they are based on the `offset` and `limit` parameters. In this case, there are two ways for you to explore the rest of the data:

- you can use the value from the next parameter to get the direct URL for the next page of requests, or
- you can build the URL for the next page from scratch using the offset and limit parameters (make sure to update the offset parameter for the request).

For the sake of learning, you will use method 2 to build the URL yourself. Then you will also compare it with the result from using the first method just to check that you created the URL correctly.

Before creating a function that will allow you to paginate, let's try to do it manually. If you compare the URLs provided by the `href` and `next` fields, you can see that while the `limit` parameter remains the same, the `offset` parameter has increased with the same value as the one stored in `limit`.

```
{
...,
'href': 'https://api.spotify.com/v1/browse/new-releases?
offset=0&limit=20',
...,
'next': 'https://api.spotify.com/v1/browse/new-releases?
offset=20&limit=20',
...
}
```

So for our next call, let's pass 20 to `offset` and keep `limit` as 20:

In [12]:
```
next_releases_response = get_new_releases(url=URL_NEW_RELEASES, access_token=tok
```

Check the values for `href` and `next` in the new response `next_releases_response`:

In [13]:
```
next_releases_response.get('albums').get('href')
```

Out[13]:
```
'https://api.spotify.com/v1/browse/new-releases?offset=20&limit=20'
```

In [14]:
```
next_releases_response.get('albums').get('next')
```

Out[14]:
```
'https://api.spotify.com/v1/browse/new-releases?offset=40&limit=20'
```

Given these results, you can see that the `offset` increases by the value of the `limit`. As the responses show that the `total` value is 100, this means that you can access the last page of responses by using an `offset` of 80, while keeping the `limit` value as 20.

In [15]:
```
last_releases_response = get_new_releases(url=URL_NEW_RELEASES, access_token=tok
```

In [16]:
```
print(last_releases_response.get('albums').get('previous'))
print(last_releases_response.get('albums').get('next'))
```

```
https://api.spotify.com/v1/browse/new-releases?offset=60&limit=20
None
```

You can see that the value of the `next` field is `None`, indicating that you reached the last page. On the other hand, you can see that `previous` contains the URL to request

the data from the previous page, so you can even go backward if required.

Exercise 2

Follow the instructions to create a new function that will handle pagination, based on the `get_new_releases` function:

1. Check the function definition, you have to provide a callable (`endpoint_request`) that corresponds to the function that performs the API call to get the new album releases.
2. Before the `while` loop, create a dictionary named `kwargs` with the following keys:
   - `'url'` : the URL to perform the call passed to the function as a parameter.
   - `'access_token'` : the access token passed to the function as a parameter.
   - `'offset'` : page's offset for the paginated request.
   - `'limit'` : maximum number of elements in the page's request.
3. Call the `endpoint_request()` function with the keyword arguments that you specified in the `kwargs` dictionary. Assign it to `response` .
4. Extend the `responses` list with the album's `items` from the `response` .
5. Create a variable `total_elements` that hosts the total number of elements from the `response` . Remember that the `response` has a field named `'albums'` that has the `'total'` number of elements. If you have any doubt about the response structure, remember to see the documentation.
6. Run the `while` loop as long as the `offset` value is smaller than `total_elements` variable you defined before.
7. Inside the `while` loop do the following steps:
   - Update the `offset` value with the current value from the request you did plus the `limit` value.
   - Repeat the definition of the `kwargs` dictionary with the same parameters. Note that in this case the `offset` value has been updated.
   - Repeat steps 3 and 4.

In [18]:
```
def paginated_new_releases(endpoint_request: Callable, url: str, access_token: s
    """Allows to perform pagination over and API request done by the endpoint_re

    Args:
        endpoint_request (Callable): Function that performs the API Calls
        url (str): Endpoint's URL for the request
        access_token (str): Access token
        offset (int, optional): Offset of the page's request. Defaults to 0.
        limit (int, optional): Limit of the page's request. Defaults to 20.

    Returns:
        list: List with the requested items
    """

    responses = []

    ### START CODE HERE ### (~ 19 lines of code)
    # Create a dictionary named kwargs with the values corresponding to the keys
    kwargs = {
```

```
            "url": url,
            "access_token": access_token,
            "offset": offset,
            "limit": limit,
        }


    # Call the endpoint_request() function with the arguments specified in the k
    response = endpoint_request(**kwargs)
    # Use extend() method to add the albums' items to the list of responses.
    responses.extend(response.get('albums').get('items'))
    # Get the total number of the elements in albums and save it in the variable
    total_elements = response.get('albums').get('total')

    # Run the loop as long as the offset value is smaller than total_elements.
    while offset < total_elements:
        # Update the offset value with the current value from the request you di
        offset = response.get('albums').get('offset') + limit
        # Repeat the definition of the kwargs dictionary with the same parameter
        kwargs = {

            "url": url,
            "access_token": access_token,
            "offset": offset,
            "limit": limit,
        }

        # Call the endpoint_request() function with the arguments specified in t
        response = endpoint_request(**kwargs)
         # Use extend() method to add the albums' items to the list of responses
        responses.extend(response.get('albums').get('items'))
    ### END CODE HERE ###

        print(f"Finished iteration for page with offset: {offset-limit}")

    return responses
```

Now, execute the `paginated_new_releases` with the function `get_new_releases` as the `endpoint_request` callable parameter. Use the same URL used in the previous `get_new_releases` call, as well as the access token. Set the initial `offset` as 0. For the limit, the default value is 20 but you can play with other values if you want.

In [19]:
```
responses = paginated_new_releases(endpoint_request=get_new_releases,
                                   url=URL_NEW_RELEASES,
                                   access_token=token.get('access_token'),
                                   offset=0, limit=20)
```

```
Finished iteration for page with offset: 0
Finished iteration for page with offset: 20
Finished iteration for page with offset: 40
Finished iteration for page with offset: 60
Finished iteration for page with offset: 80
```

Expected Output

```
Finished iteration for page with offset: 0
Finished iteration for page with offset: 20
Finished iteration for page with offset: 40
```

```
Finished iteration for page with offset: 60
Finished iteration for page with offset: 80
```

Have a look at one of the item:

In [20]:
```
responses[0]
```

Out[20]:

```
{'album_type': 'album',
 'artists': [{'external_urls': {'spotify': 'https://open.spotify.com/artist/06H
L4z0CvFAxyc27GXpf02'},
   'href': 'https://api.spotify.com/v1/artists/06HL4z0CvFAxyc27GXpf02',
   'id': '06HL4z0CvFAxyc27GXpf02',
   'name': 'Taylor Swift',
   'type': 'artist',
   'uri': 'spotify:artist:06HL4z0CvFAxyc27GXpf02'}],
 'available_markets': ['AR',
  'AU',
  'AT',
  'BE',
  'BO',
  'BR',
  'BG',
  'CA',
  'CL',
  'CO',
  'CR',
  'CY',
  'CZ',
  'DK',
  'DO',
  'DE',
  'EC',
  'EE',
  'SV',
  'FI',
  'FR',
  'GR',
  'GT',
  'HN',
  'HK',
  'HU',
  'IS',
  'IE',
  'IT',
  'LV',
  'LT',
  'LU',
  'MY',
  'MT',
  'MX',
  'NL',
  'NZ',
  'NI',
  'NO',
  'PA',
  'PY',
  'PE',
  'PH',
  'PL',
  'PT',
  'SG',
  'SK',
  'ES',
  'SE',
  'CH',
  'TW',
  'TR',
```

```
'UY',
'US',
'GB',
'AD',
'LI',
'MC',
'ID',
'JP',
'TH',
'VN',
'RO',
'IL',
'ZA',
'SA',
'AE',
'BH',
'QA',
'OM',
'KW',
'EG',
'MA',
'DZ',
'TN',
'LB',
'JO',
'PS',
'IN',
'KZ',
'MD',
'UA',
'AL',
'BA',
'HR',
'ME',
'MK',
'RS',
'SI',
'KR',
'BD',
'PK',
'LK',
'GH',
'KE',
'NG',
'TZ',
'UG',
'AG',
'AM',
'BS',
'BB',
'BZ',
'BT',
'BW',
'BF',
'CV',
'CW',
'DM',
'FJ',
'GM',
'GE',
```

```
'GD',
'GW',
'GY',
'HT',
'JM',
'KI',
'LS',
'LR',
'MW',
'MV',
'ML',
'MH',
'FM',
'NA',
'NR',
'NE',
'PW',
'PG',
'WS',
'SM',
'ST',
'SN',
'SC',
'SL',
'SB',
'KN',
'LC',
'VC',
'SR',
'TL',
'TO',
'TT',
'TV',
'VU',
'AZ',
'BN',
'BI',
'KH',
'CM',
'TD',
'KM',
'GQ',
'SZ',
'GA',
'GN',
'KG',
'LA',
'MO',
'MR',
'MN',
'NP',
'RW',
'TG',
'UZ',
'ZW',
'BJ',
'MG',
'MU',
'MZ',
'AO',
```

```
  'CI',
  'DJ',
  'ZM',
  'CD',
  'CG',
  'IQ',
  'LY',
  'TJ',
  'VE',
  'ET',
  'XK'],
 'external_urls': {'spotify': 'https://open.spotify.com/album/1Mo4aZ8pdj6L1jx8z
SwJnt'},
 'href': 'https://api.spotify.com/v1/albums/1Mo4aZ8pdj6L1jx8zSwJnt',
 'id': '1Mo4aZ8pdj6L1jx8zSwJnt',
 'images': [{'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e025076e4160d018e378f488c33',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d000048515076e4160d018e378f488c33',
   'width': 64},
  {'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b2735076e4160d018e378f488c33',
   'width': 640}],
 'name': 'THE TORTURED POETS DEPARTMENT',
 'release_date': '2024-04-18',
 'release_date_precision': 'day',
 'total_tracks': 16,
 'type': 'album',
 'uri': 'spotify:album:1Mo4aZ8pdj6L1jx8zSwJnt'}
```


You can check the `responses` variable to see if all the elements were downloaded successfully.

In [21]:
```
len(responses)
```

Out[21]:
```
100
```

With the `paginated_new_releases` function that you created, you are now able to get all 100 available items.

Exercise 3

The function `get_new_releases` can handle pagination by passing the `offset` and `limit` parameters or only using the `next` parameter. Create another function that uses the `next` parameter to perform the pagination and compare your results from the previous exercise. Follow the instructions below:

The dictionary `kwargs` is now defined with the following keys:

- `'url'` : the URL to perform the call passed to the function as a parameter.
- `'access_token'` : the access token passed to the function as a parameter.
- `'next'` : the URL to generate the next request, defined as an empty string for the first call.

Inside the `while` loop:

1. Call the `endpoint_request()` function with the keyword arguments that you specified in the `kwargs` dictionary. Assign it to `response`.
2. Extend the `responses` list with the albums' `items` from the `response`.
3. Reassign the value of `next_page` as the `'next'` value from the `response["albums"]` dictionary. If you have any doubt about the response structure, remember to see the documentation.
4. Update the `kwargs` dictionary: set the value of the key `'next'` as the variable `next_page`.

In [23]:
```
def paginated_with_next_new_releases(endpoint_request: Callable, url: str, acces
    """Manages pagination for API requests done with the endpoint_request callab

    Args:
        endpoint_request (Callable): Function that performs API request
        url (str): Base URL for the request
        access_token (str): Access token

    Returns:
        list: Responses stored in a list
    """
    responses = []

    next_page = url

    kwargs = {
            "url": url,
            "access_token": access_token,
            "next": ""
        }

    while next_page:

        ### START CODE HERE ### (~ 4 lines of code)
        # Call the endpoint_request() function with the arguments specified in t
        response = endpoint_request(**kwargs)
        # Use extend() method to add the albums' items to the list of responses.
        responses.extend(response.get('albums').get('items'))
        # Reassign the value of next_page as the 'next' value from the response[
        next_page = response.get('albums').get('next')
        # Update the kwargs dictionary: set the value of the key 'next' as the v
        kwargs["next"] = next_page
        ### END CODE HERE ###

        print(f"Executed request with URL: {response.get('albums').get('href')}.

    return responses
```

Now, perform the new paginated call:

In [24]:
```
responses_with_next = paginated_with_next_new_releases(endpoint_request=get_new_
                                                        url=URL_NEW_RELEASE
                                                        access_token=token.
```

```
Executed request with URL: https://api.spotify.com/v1/browse/new-releases?offset=
0&limit=20.
Executed request with URL: https://api.spotify.com/v1/browse/new-releases?offset=
20&limit=20.
Executed request with URL: https://api.spotify.com/v1/browse/new-releases?offset=
40&limit=20.
Executed request with URL: https://api.spotify.com/v1/browse/new-releases?offset=
60&limit=20.
Executed request with URL: https://api.spotify.com/v1/browse/new-releases?offset=
80&limit=20.
```

Have a look at one of the responses:

In [25]:
```
responses_with_next[0]
```

Out[25]:

```
{'album_type': 'album',
 'artists': [{'external_urls': {'spotify': 'https://open.spotify.com/artist/06H
L4z0CvFAxyc27GXpf02'},
   'href': 'https://api.spotify.com/v1/artists/06HL4z0CvFAxyc27GXpf02',
   'id': '06HL4z0CvFAxyc27GXpf02',
   'name': 'Taylor Swift',
   'type': 'artist',
   'uri': 'spotify:artist:06HL4z0CvFAxyc27GXpf02'}],
 'available_markets': ['AR',
  'AU',
  'AT',
  'BE',
  'BO',
  'BR',
  'BG',
  'CA',
  'CL',
  'CO',
  'CR',
  'CY',
  'CZ',
  'DK',
  'DO',
  'DE',
  'EC',
  'EE',
  'SV',
  'FI',
  'FR',
  'GR',
  'GT',
  'HN',
  'HK',
  'HU',
  'IS',
  'IE',
  'IT',
  'LV',
  'LT',
  'LU',
  'MY',
  'MT',
  'MX',
  'NL',
  'NZ',
  'NI',
  'NO',
  'PA',
  'PY',
  'PE',
  'PH',
  'PL',
  'PT',
  'SG',
  'SK',
  'ES',
  'SE',
  'CH',
  'TW',
  'TR',
```

```
'UY',
'US',
'GB',
'AD',
'LI',
'MC',
'ID',
'JP',
'TH',
'VN',
'RO',
'IL',
'ZA',
'SA',
'AE',
'BH',
'QA',
'OM',
'KW',
'EG',
'MA',
'DZ',
'TN',
'LB',
'JO',
'PS',
'IN',
'KZ',
'MD',
'UA',
'AL',
'BA',
'HR',
'ME',
'MK',
'RS',
'SI',
'KR',
'BD',
'PK',
'LK',
'GH',
'KE',
'NG',
'TZ',
'UG',
'AG',
'AM',
'BS',
'BB',
'BZ',
'BT',
'BW',
'BF',
'CV',
'CW',
'DM',
'FJ',
'GM',
'GE',
```

```
'GD',
'GW',
'GY',
'HT',
'JM',
'KI',
'LS',
'LR',
'MW',
'MV',
'ML',
'MH',
'FM',
'NA',
'NR',
'NE',
'PW',
'PG',
'WS',
'SM',
'ST',
'SN',
'SC',
'SL',
'SB',
'KN',
'LC',
'VC',
'SR',
'TL',
'TO',
'TT',
'TV',
'VU',
'AZ',
'BN',
'BI',
'KH',
'CM',
'TD',
'KM',
'GQ',
'SZ',
'GA',
'GN',
'KG',
'LA',
'MO',
'MR',
'MN',
'NP',
'RW',
'TG',
'UZ',
'ZW',
'BJ',
'MG',
'MU',
'MZ',
'AO',
```

```
  'CI',
  'DJ',
  'ZM',
  'CD',
  'CG',
  'IQ',
  'LY',
  'TJ',
  'VE',
  'ET',
  'XK'],
 'external_urls': {'spotify': 'https://open.spotify.com/album/1Mo4aZ8pdj6L1jx8z
SwJnt'},
 'href': 'https://api.spotify.com/v1/albums/1Mo4aZ8pdj6L1jx8zSwJnt',
 'id': '1Mo4aZ8pdj6L1jx8zSwJnt',
 'images': [{'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e025076e4160d018e378f488c33',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d000048515076e4160d018e378f488c33',
   'width': 64},
  {'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b2735076e4160d018e378f488c33',
   'width': 640}],
 'name': 'THE TORTURED POETS DEPARTMENT',
 'release_date': '2024-04-18',
 'release_date_precision': 'day',
 'total_tracks': 16,
 'type': 'album',
 'uri': 'spotify:album:1Mo4aZ8pdj6L1jx8zSwJnt'}
```


2.4 - Optional - API Rate Limits

Note: This is an optional section.

Another important aspect to take into account when working with APIs is regarding the rate limits. Rate limiting is a mechanism used by APIs to control the number of requests that a client can make within a specified period of time. It helps prevent abuse or overload of the API by limiting the frequency or volume of requests from a single client. Here's how rate limiting typically works:

- Request Quotas: APIs may enforce a maximum number of requests that a client can make within a given time window, for example, 100 requests per minute.
- Time Windows: The time window specifies the duration over which the request quota is measured. For example, a rate limit of 100 requests per minute means that the client can make up to 100 requests in any 60-second period.
- Response to Exceeding Limits: When a client exceeds the rate limit, the API typically responds with an error code (such as 429 Too Many Requests) or a message indicating that the rate limit has been exceeded. This allows clients to adjust their behavior accordingly, such as by implementing exponential backoff and other retry strategies. (Check here or here).

- Rate Limit Headers: APIs may include headers in the response to indicate the client's current rate limit status, such as the number of requests remaining until the limit resets or the time at which the limit will reset.

Rate limiting helps maintain the stability and reliability of APIs by ensuring fair access to resources and protecting against abusive or malicious behavior. It also allows API providers to allocate resources more effectively and manage traffic loads more efficiently.

You can also see more of the specifics of the rate limits of the Spotify Web API in the documentation. Particularly, this API doesn't enforce a hard limit for the number of requests done but it works dynamically based on the number of calls within a rolling 30 seconds window. You can find some blogs where experiments have been done to identify the average number of requests per minute.

Below you are provided with a code that benchmarks the API calls; you can play with the number of requests and the request interval to see the average time of a request. In case you perform too many requests so that you violate the rate limits, you will get a 429 status code.

Note: This code may take a few minutes to run.

In [27]:
```
import time

# Define the Spotify API endpoint
endpoint = 'https://api.spotify.com/v1/browse/new-releases'

headers = get_auth_header(access_token=token.get('access_token'))

# Define the number of requests to make
num_requests = 200

# Define the interval between requests (in seconds)
request_interval = 0.1  # Adjust as needed based on the API rate limit

# Store the timestamps of successful requests
success_timestamps = []

# Make repeated requests to the endpoint
for i in range(num_requests):
    # Make the request
    response = requests.get(url=endpoint, headers=headers)

    # Check if the request was successful
    if response.status_code == 200:
        success_timestamps.append(time.time())
    else:
        print(f'Request {i+1}: Failed with code {response.status_code}')

    # Wait for the specified interval before making the next request
    time.sleep(request_interval)

# Calculate the time between successful requests
if len(success_timestamps) > 1:
    time_gaps = [success_timestamps[i] - success_timestamps[i-1] for i in range(
    print(f'Average time between successful requests: {sum(time_gaps) / len(time
```

```
else:
    print('At least two successful requests are needed to calculate the time bet
```

```
Average time between successful requests: 0.40 seconds
```

3 - Batch pipeline

Now that you have learned the basics of working with APIs, let's create a pipeline that extracts the track information for the new released albums. For that, you will use two endpoints:

- The same Get New Releases endpoint you used in the previous exercises.
- The Get Album Tracks endpoint. This endpoint allows you to get Spotify catalog information about an album's tracks.

In the `src/` folder, you are given three scripts (`authentication.py` , `endpoint.py` and `main.py`) that will allow you to perform such extraction.

- The `endpoint.py` file contains two paginated api calls. The first one `get_paginated_new_releases` allows you get the list of new album releases using the same paginated call you used in the first part. The second one `get_paginated_album_tracks` allows you to get Spotify catalog information about an album's tracks using the Get Album Tracks endpoint.
- The `authentication.py` file contains the script of the `get_token` function that returns an access token.
- The `main.py` file calls the first paginated API call to get the ids of the new albums. Then for each album id, the second paginated API call is performed to extract the catalog information for each album id.

At this moment, the code manages paginated requests but we haven't taken into account that our access token has a limited time, so if your pipeline requests last more than 3600 seconds, you can get a 401 status code error. So the first step is to write a routine that handles token refresh in the `get_paginated_new_releases` . Follow the instructions to implement this routine.

Exercise 4

Open the file located at `src/endpoint.py` .

Search for the `get_paginated_new_releases` function. Create an if condition over the `response.status_code` and compare it with the value 401. This means that if the returned status code is 401 (Unauthorized) you will perform the next steps:

- Use the `kwargs` argument that is passed to the `get_paginated_new_releases` function; pass it to the `get_token` callable and assign it to a variable named `token_response` .
- Create an internal condition in which you will check if the key `"access_token"` is

in the `token_response` dictionary. If true, you will call the `get_auth_header` function with the corresponding access token and assign the result to the variable `headers`. Note the usage of the `continue` keyword to make sure that the request will be executed again.
- If the condition over `"access_token"` is false, just return an empty list.

Save changes in the file `src/endpoint.py`.

Now that we know how to refresh the token in case it expires, it's time to continue with the rest of the code. Now that we have the new album releases, the idea is to extract the information of the tracks that compose each album.

To get this information, you will use the Get Album Tracks endpoint. Take a moment to read the documentation and understand how to request data from this particular endpoint.

Open the file at `src/main.py`. There you will see after the call to the `get_paginated_new_releases` function that you are extracting the albums' IDs from the response and saving them into the `albums_ids` list. Those IDs will be used in the request. Also, search for the following constant:

- `URL_ALBUM_TRACKS`: The base URL to get information from a particular album. Take a look at the documentation, you can see that you will have to complement that URL with the album ID and with the `tracks` string to complete the endpoint.

This information will be passed to the function `get_paginated_album_tracks` to construct the full endpoint for the API call. In the next exercise, you will work on completing this function in the file `src/endpoint.py`.

Exercise 5

Go back to the `src/endpoint.py` file. Search for the comment `Exercise 5` and follow the instructions to complete the function `get_paginated_album_tracks`.

1. The function blueprint for `get_paginated_album_tracks` is already provided to you. The first thing you have to do is to call the `get_auth_header` function with the access token and pass it to a variable `headers`.
2. Create the `requests_url` by using the `base_url` and `album_id` parameters. Notice that the `tracks` are added to the URL endpoint.
3. In the `while` loop, perform a GET request using the `request_url` and `headers` that you created in the previous step. Assign the result to `response`.
4. Implement the same token refresh routine as before for the `get_paginated_new_releases` function.
5. After the token refresh, convert the `response` to JSON using the `.json()` method. Assign it to `response_json`.
6. Extend the `album_data` list with the value from `"items"` in `response_json`.
7. Update `request_url` with the `"next"` value from `response_json`.

Save changes in the file `src/endpoint.py`.

Exercise 6

Go back to the `src/main.py` file. Search for the comment `Exercise 6`. Inside the loop that iterates through the `albums_ids`, there's a call for the `get_paginated_album_tracks` function. Follow the instructions to define the following parameters for the given function:

- `base_url`: Use the `URL_ALBUM_TRACKS` constant defined for you.
- `access_token`: Make sure to pass the `"access_token"` from the `token` object.
- `album_id`.
- `get_token`: pass the same `get_token` function.
- Finally, pass the `kwargs` dictionary defined at the start of the `main()` function.

The function response will be assigned to the variable `album_data`.

Save changes in the file `src/main.py`.

Inside the same for loop, the response `album_data` is added to the `album_items` dictionary, using the corresponding album id as key. Finally, after iterating through all albums, you can see that the dictionary `album_items` is saved to a JSON file in the local environment. Take a look at the format of the filename, which takes into account the current date time to avoid collision with other files.

Run the following commands in the terminal to run the `main.py` script:

```
cd src
python main.py
```

Notes: To open the terminal, click on Terminal -> New Terminal in the menu:

Once the script is finished, you should be able to see a file named `album_items_<DATETIME>.json` in the folder `src`.

4 - Optional - Spotipy SDK

In several cases, the API developers also provide a Software Development Kit (SDK) to connect and perform requests to the different endpoints of the API without the necessity of creating the code from scratch. For Spotify Web API they developed the Spotipy SDK to do it. Let's see an example of how it will work to replicate the extraction of data from the new album releases endpoint in a paginated way.

In [28]:
```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
```

In [29]:
```
credentials = SpotifyClientCredentials(
        client_id=CLIENT_ID, client_secret=CLIENT_SECRET
    )

spotify = spotipy.Spotify(client_credentials_manager=credentials)
```

You can see that the `credentials` object handles the authentication process and

contains the token to be used in later requests.

Note: Please ignore the `DeprecationWarning` message if you see an access token in the output.

In [30]:
```
credentials.get_access_token()
```

```
/tmp/ipykernel_1714/2682040240.py:1: DeprecationWarning: You're using 'as_dict =
True'.get_access_token will return the token string directly in future versions.
Please adjust your code accordingly, or use get_cached_token instead.
  credentials.get_access_token()
```

Out[30]:
```
{'access_token': 'BQAzHpVWJih7P8lUeBFNvIrU0_mBn7TxH5L0-rwCKKBDY10aLfCaLtlXtvFuE
dnmbYXD4pIV4QuYmCRGkFS8ZtrlmX_zIbekXexHqt3W8x6Qrbbz4gVKvRWyRE1ibE25LN-3tCwwsm
o',
 'token_type': 'Bearer',
 'expires_in': 3600,
 'expires_at': 1752043098}
```

Let's get data from of the new album releases, as you did in the previous example:

In [31]:
```
limit = 20
response = spotify.new_releases(limit=limit)
```

You can also paginate through these responses. If you check the documentation of the `new_releases` method, you can see that you can specify the parameter `offset`, as you previously did.

Exercise 7

Let's perform a paginated request with the SDK. We can then check the results of this paginated request against the total number of elements we saw in the previous exercises, which would be 100. Follow the instructions to finish the code to perform paginated requests.

1. Extend the `album_data` list with the `items` from the `albums` key in the previous `response`.
2. Get the `total` number of elements from the response and assign it to the variable `total_albums_elements`.
3. Create a list of offset indexes. As you already have the data from the first call, your starting offset index should be the `limit` value that you used to make the first request. The list should finish at `total_albums_elements` and the pace should be the same `limit` value. Save it into `offset_idx`.
4. Start the pagination: iterate over each index in `offset_idx`. In `response_page`, assign the response from the request to the `new_releases` method using the corresponding offset index and limit.
5. Extend again the `album_data` with the album `items` that you get in the `response_page`.

You can visually inspect the values in the `album_data` and make sure the list length is the same as the total number of elements that are available to request.

In [32]:
```
def paginated_new_releases_sdk(limit: int=20) -> list:

    album_data = []
    ### START CODE HERE ### (~ 6 lines of code) spotify.new_releases(limit=limit
    album_data.extend(response.get('albums', {}).get('items', []))
    total_albums_elements = response['albums']['total']
    offset_idx = list(range(limit, total_albums_elements, limit))

    for idx in offset_idx:

        response_page = spotify.new_releases(limit=limit, offset=idx)
        album_data.extend(response_page.get('albums', {}).get('items', []))
    ### END CODE HERE ###
    return album_data

album_data_sdk = paginated_new_releases_sdk()
album_data_sdk[0]
```

Out[32]:

```
{'album_type': 'album',
 'artists': [{'external_urls': {'spotify': 'https://open.spotify.com/artist/06H
L4z0CvFAxyc27GXpf02'},
   'href': 'https://api.spotify.com/v1/artists/06HL4z0CvFAxyc27GXpf02',
   'id': '06HL4z0CvFAxyc27GXpf02',
   'name': 'Taylor Swift',
   'type': 'artist',
   'uri': 'spotify:artist:06HL4z0CvFAxyc27GXpf02'}],
 'available_markets': ['AR',
  'AU',
  'AT',
  'BE',
  'BO',
  'BR',
  'BG',
  'CA',
  'CL',
  'CO',
  'CR',
  'CY',
  'CZ',
  'DK',
  'DO',
  'DE',
  'EC',
  'EE',
  'SV',
  'FI',
  'FR',
  'GR',
  'GT',
  'HN',
  'HK',
  'HU',
  'IS',
  'IE',
  'IT',
  'LV',
  'LT',
  'LU',
  'MY',
  'MT',
  'MX',
  'NL',
  'NZ',
  'NI',
  'NO',
  'PA',
  'PY',
  'PE',
  'PH',
  'PL',
  'PT',
  'SG',
  'SK',
  'ES',
  'SE',
  'CH',
  'TW',
  'TR',
```

```
'UY',
'US',
'GB',
'AD',
'LI',
'MC',
'ID',
'JP',
'TH',
'VN',
'RO',
'IL',
'ZA',
'SA',
'AE',
'BH',
'QA',
'OM',
'KW',
'EG',
'MA',
'DZ',
'TN',
'LB',
'JO',
'PS',
'IN',
'KZ',
'MD',
'UA',
'AL',
'BA',
'HR',
'ME',
'MK',
'RS',
'SI',
'KR',
'BD',
'PK',
'LK',
'GH',
'KE',
'NG',
'TZ',
'UG',
'AG',
'AM',
'BS',
'BB',
'BZ',
'BT',
'BW',
'BF',
'CV',
'CW',
'DM',
'FJ',
'GM',
'GE',
```

```
'GD',
'GW',
'GY',
'HT',
'JM',
'KI',
'LS',
'LR',
'MW',
'MV',
'ML',
'MH',
'FM',
'NA',
'NR',
'NE',
'PW',
'PG',
'WS',
'SM',
'ST',
'SN',
'SC',
'SL',
'SB',
'KN',
'LC',
'VC',
'SR',
'TL',
'TO',
'TT',
'TV',
'VU',
'AZ',
'BN',
'BI',
'KH',
'CM',
'TD',
'KM',
'GQ',
'SZ',
'GA',
'GN',
'KG',
'LA',
'MO',
'MR',
'MN',
'NP',
'RW',
'TG',
'UZ',
'ZW',
'BJ',
'MG',
'MU',
'MZ',
'AO',
```

```
  'CI',
  'DJ',
  'ZM',
  'CD',
  'CG',
  'IQ',
  'LY',
  'TJ',
  'VE',
  'ET',
  'XK'],
 'external_urls': {'spotify': 'https://open.spotify.com/album/1Mo4aZ8pdj6L1jx8z
SwJnt'},
 'href': 'https://api.spotify.com/v1/albums/1Mo4aZ8pdj6L1jx8zSwJnt',
 'id': '1Mo4aZ8pdj6L1jx8zSwJnt',
 'images': [{'height': 300,
   'url': 'https://i.scdn.co/image/ab67616d00001e025076e4160d018e378f488c33',
   'width': 300},
  {'height': 64,
   'url': 'https://i.scdn.co/image/ab67616d000048515076e4160d018e378f488c33',
   'width': 64},
  {'height': 640,
   'url': 'https://i.scdn.co/image/ab67616d0000b2735076e4160d018e378f488c33',
   'width': 640}],
 'name': 'THE TORTURED POETS DEPARTMENT',
 'release_date': '2024-04-18',
 'release_date_precision': 'day',
 'total_tracks': 16,
 'type': 'album',
 'uri': 'spotify:album:1Mo4aZ8pdj6L1jx8zSwJnt'}
```

In [33]:
```
len(album_data_sdk)
```

Out[33]:
```
100
```

In this lab you learned the basics of ingesting data from the API. You worked with authentication process and pagination in a manual way as well as using an API SDK.

In [ ]:
```

```