Create a playlist of the happiest songs of your favourite artists

17.05.2017

Working with Spotify Web API

By Magdalena Sendal

Agenda

1. About me
2. Introduction to Web APIs and JSON
3. Look at Spotify Web API documentation
4. Practice accessing API in Python

What we will learn today?

- Use Python requests library to access data via web APIs
- Parse JSON data

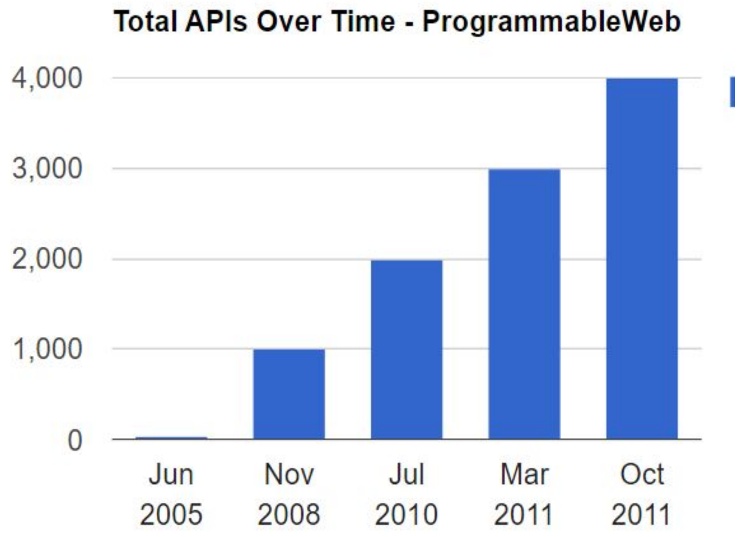
Presenters

Magda:

- From Wroclaw, Poland
- Study: Computer Science
- Work in BI, Data-Warehousing, Reporting Automation
- Data-Warehouse Engineer @foodora:
  - Python for ETL
  - Work with web APIs to fetch data from Google, Facebook, Zendesk …

Why this topic?

- API is exponential increase in popularity and usage
- important tool for web developers
- marketing tool
- have fun building “mushups”



WEB API (Application Programming Interface)

API is like user interface, just with different users in mind

- communication interface that uses HTTP protocol and JSON or XML as data exchange format
- allows communication between users and the system located on the server to access the site's resources
- Examples of web services with open APIs:
  - Google, Facebook, Twitter, Yelp, Spotify
  - More in online api library

Spotify WEB Api

- Example - let’s compare :

  https://api.spotify.com/v1/search?q=radiohead&type=artist

  https://open.spotify.com/search/artists/radiohead

- Developers documentation
  - https://developer.spotify.com/web-api/
- Endpoints
  - Which URL needs to be called to get specific resources
- Requests parameters
  - Either as path element or parameter
- Response format

JSON (JavaScript Object Notation)

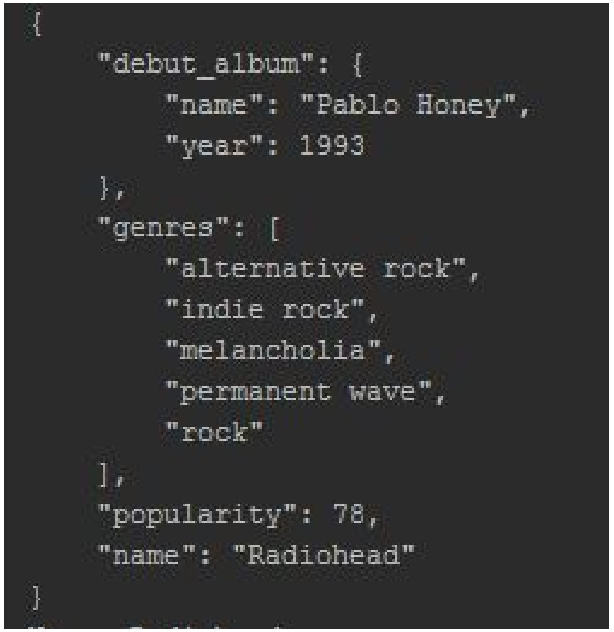
- Invented in 2001, as alternative to XML
- Data is serialized in nested “lists” and “dictionaries” (key- value pairs)
- Came from JavaScript, but is language independent
- The world's best-loved data interchange format

Python:

- json - JSON encoder and decoder

Examples - extracting data from json :

- Json1.py
- Json2.py



```
{
    "debut_album": {
        "name": "Pablo Honey",
        "year": 1993
    },
    "genres": [
        "alternative rock",
        "indie rock",
        "melancholia",
        "permanent wave",
        "rock"
    ],
    "popularity": 78,
    "name": "Radiohead"
}
```

Json Exercise

1. List all the album (not signles) titles and release dates of a selected artist
   https://developer.spotify.com/web-api/get-artists-albums/

   a. Create a URL by passing the id of your selected artist
   b. Store the result in json file
   c. Load the file and deserialize to python object
   d. Parse the album titles

Json Excercise - Solution

```
https://api.spotify.com/v1/artists/4Z8W4fKeB5YxbusRsdQVPb/albums?album_type=album
```

- How many albums did you get in your response?

Accessing Api with Python

- Instead of storing results to the file we want to fetch the data with Python :
  - requests

Requests allows you to send *organic, grass-fed* HTTP/1.1 requests, without the need for manual labor. There's no need to manually add query strings to your URLs, or to form-encode your POST data

- Example:
  - requests1.py
  - requests2.py

Http Requests

Request message from client to server consists of:

1. Method e.g.:
   - GET - used to fetch the data
   - POST - used to send the data to web server

2. Request-URI

   - address where the resource sits

3. Request Header Fields:

- allow the client to pass additional information about the request, and about the client itself, to the server

Requests Excercise

- Write a function get_album_ids
    - Takes artist name as input
    - Returns list of album_ids
    - Use previous function search_artist_id and pass the artist id to the url

Pagination

- When response to your request exceeds the limit of items per page:
  - https://developer.spotify.com/web-api/user-guide/#pagination
  - Extending limit
  - Comparing current offset vs total number of items
  - Parse und use the URL to the next page of items

- Example:
  - pagination.py

Steps to create a playlist

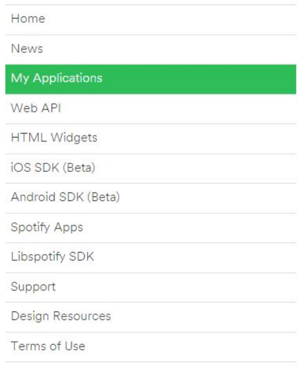
1. Search for an artist and get its id
2. Get all albums ids of an artist
3. Get all the songs for each album (id)
4. Get audio features for all tracks (id)
5. Sort tracks using valence
6. Create a new playlist
7. Add songs with highest valence score to your new playlist

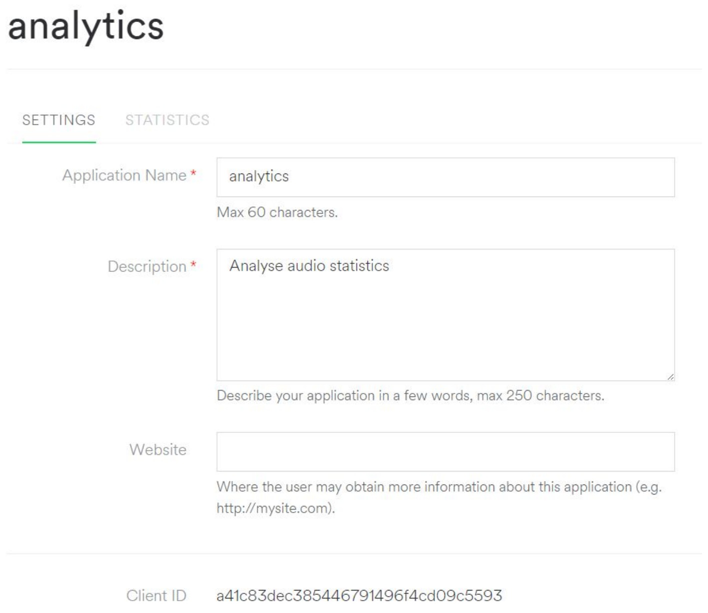
AUTHORIZING YOUR ACCOUNT

We must authorize to access some resources (e.g. track audio features)

First, you need to register your app:

1. Go to My Applications (make sure you are logged in first)
2. Create an App
3. Choose any name and description for your application
4. In Redirect_URI please specify https://127.0.0.1:8000
5. Save your client id and client token in credentials.py





analytics

SETTINGS STATISTICS

Application Name * analytics
Max 60 characters.

Description * Analyse audio statistics
Describe your application in a few words, max 250 characters.

Website
Where the user may obtain more information about this application (e.g. http://mysite.com).

Client ID a41c83dec385446791496f4cd09c5593

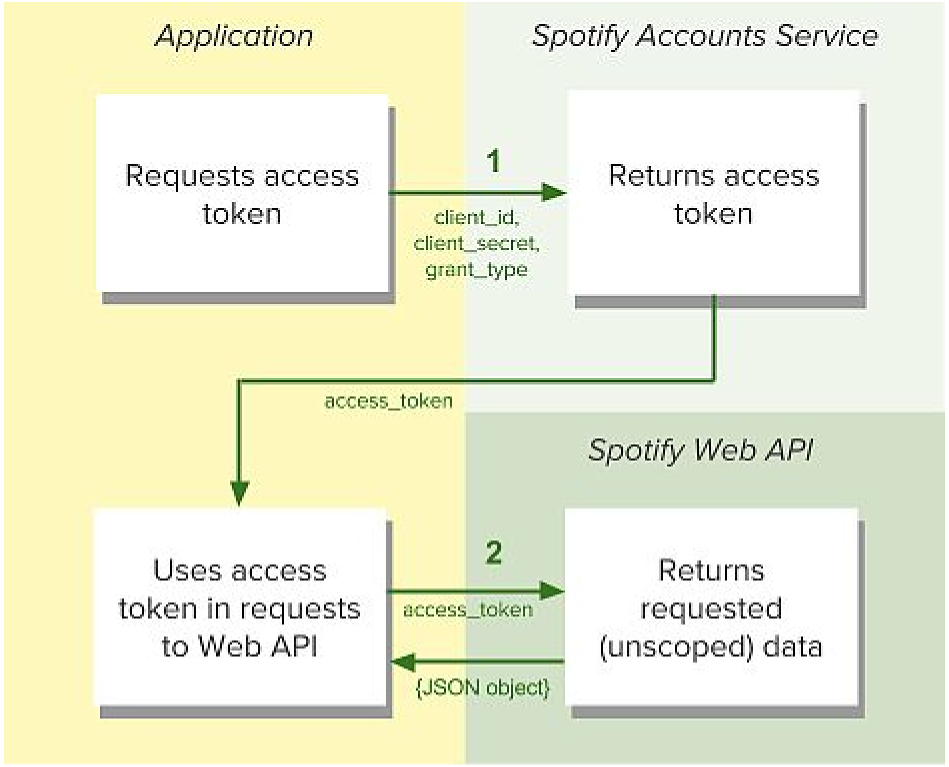
AUTHORIZING YOUR ACCOUNT - CLIENT CREDENTIALS FLOW

1. Having client_id and client_secret make a POST call to `https://accounts.spotify.com/api/token`

    1. Set the client details in ‘auth’ variable of request.post
    2. Set grant_type in ‘data’ variable

2. Use the access token to access the Spotify Web API

Example : auth.py



Client Credentials Flow

Quantifying positiveness of a song

- Knowing how to authorize your app we can now use get-several-audio-features endpoint
- Use ‘valence’ to quantify musical positiveness of a track:
  - Maeasure from 0 to 1. The higher valence to more positive the song is
- All we need to do, is to sort the songs by valence!
- Exercise:
  - Extend code from requests3.py to add :
    - Function get_auth() that returns your auth token
    - Function get_tracks_valence(track_ids) that returns dictionary where keys are track_ids and values are valence

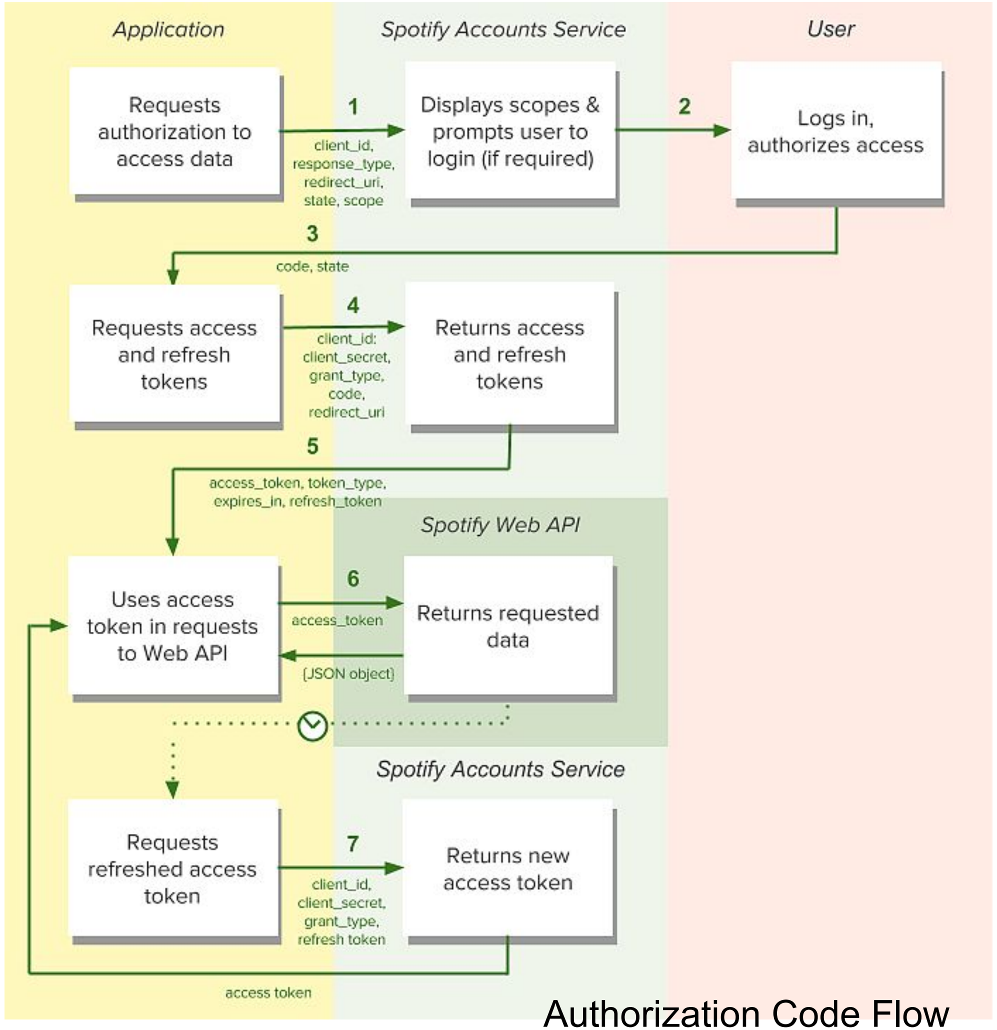
Final Step- Create a playlist

1. Create a new playlist using create-playlist endpoint
2. Add tracks using add-tracks-to-playlist/

Because we are touching user profile our application needs to follow full Authorization Code Flow and grant our application ‘playlist-modify-private’

Example:

1. In separate console run python3 -m server.http and check 127.0.0.1:8000 (Python2: python -m SimpleHttpServer)
2. Run auth2.py
3. Copy & paste redirected URL with code


Application
Spotify Accounts Service
User
Requests authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Displays scopes & prompts user to login (if required)
2
Logs in, authorizes access
3
code, state
Requests access and refresh tokens
4
client_id: client_secret, grant_type, code, redirect_uri
Returns access and refresh tokens
5
access_token, token_type, expires_in, refresh_token
Spotify Web API
Uses access token in requests to Web API
6
access_token
Returns requested data
{JSON object}
Spotify Accounts Service
Requests refreshed access token
7
client_id, client_secret, grant_type, refresh token
Returns new access token
access token

Authorization Code Flow

LET'S PUT IT ALL TOGETHER

`final.py`

References and further reading

- Coursera Using Python to Access Web Data
- Reference for Spotify Web API
- Similar tutorial in R fitteR happieR
- API university on programmableweb.com