spoffy

Release 0.1.0

Jun 18, 2020

Contents

The IDE friendly sync and async Spotify API wrapper for python.

Features include:

- Typed response models
- Async AND sync support
- Built in support for aiohttp and requests (or bring your own http library)
- Support for client credentials and OAuth2 authorization flows

CHAPTER 1

Type hints everywhere

All API json responses are unmarshalled to typed python objects. (e.g. `Playlist`) This gives you excellent code completion in your IDE and makes your code less prone to simple mistakes.

Spoffy is still a thin API wrapper and the response bodies are not reshaped or restructured in any way. You get exactly the data returned from Spotify.

Note: Spotify sometimes includes undocumented attributes in responses, these are also included as attributes on the response object but they're not typed or documented.

CHAPTER 2

Sansio

Spoffy is built with sansio principles. In this case that means the choice of underlying http library is merely an implementation detail. The client can be easily implemented to work with any sync or async http client library. Out of the box, spoffy ships with a sync implementation using requests and an async implementation using aiohttp

CHAPTER 3

Usage

The Spotify wrapper comes in two flavours `SyncSpotify` to use with a synchronous http library such as requests and `AsyncSpotify` to use async/await keywords with an async http library such as aiohttp. The wrapper takes in a client as an argument, this is the part that generates http requests and sends them to the spotify api. To implement your own client use the `AsyncClient` or `SyncClient` as a base respectively, or use the one of the built in clients: `RequestsClient` or `AioHttpClient`

This is how you instantiate the spotify wrapper with the requests client:

```
from spoffy import SyncSpotify
from spoffy.io.requests import RequestsClient

spotify = SyncSpotify(RequestsClient(...))

# Or use the factory function, this is equivalent

from spoffy.io.requests import make_spotify

spotify = make_spotify(...)
```

This is how you use it with the aiohttp client:

```
import aiohttp
from spoffy import AsyncSpotify
from spoffy.io.aiohttp import AioHttpclient

...

async with aiohttp.ClientSession() as session:
    spotify = SyncSpotify(AioHttpClient(session=session))

# Or use the factory function, this is equivalent

from spoffy.io.aiohttp import make_spotify
```

(continues on next page)

(continued from previous page)

```
async with aiohttp.ClientSession() as session:
    spotify = make_spotify(session=session)
```

CHAPTER 4

Quickstart

To start using the API you must first create a client ID and client secret by going to https://developer.spotify.com/dashboard/applications , log in with your spotify account and click "Create a client ID" and go through the steps to register your app.

Once you have your app set up you can access the public API using client credentials authentication.

Note: Spoffy does not have dependencies on any http client libraries, so you must install your preferred library yourself

4.1 Use with requests

```
pip install requests
```

```
import os
import requests

from spoffy.io.requests import make_spotify


CLIENT_ID = os.environ["SPOTIFY_CLIENT_ID"]
CLIENT_SECRET = os.environ["SPOTIFY_CLIENT_SECRET"]


session = requests.Session()  # Optional

spotify = make_spotify(
    session=session, client_id=CLIENT_ID, client_secret=CLIENT_SECRET
)

spotify.auth.authorize_client()  # Client credentials auth

result = spotify.search.artists("Tom Waits")
```

(continues on next page)

(continued from previous page)

```
print("Tom Waits is this popular:", result.artists.items[0].popularity)
```

4.2 Use with AIOHTTP

```
pip install aiohttp
```

```
import os
import asyncio

import aiohttp
from spoffy.io.aiohttp import make_spotify
from spoffy.models import Artist


CLIENT_ID = os.environ["SPOTIFY_CLIENT_ID"]
CLIENT_SECRET = os.environ["SPOTIFY_CLIENT_SECRET"]


async def get_artist(name: str) -> Artist:
    async with aiohttp.ClientSession() as session:
        spotify = make_spotify(
            session=session, client_id=CLIENT_ID, client_secret=CLIENT_SECRET
        )
        await spotify.auth.authorize_client()
        result = await spotify.search.artists("Tom Waits")
        return result.artists.items[0]


loop = asyncio.get_event_loop()

data = loop.run_until_complete(get_artist("Tom Waits"))
print("Tom waits is this popular:", data.popularity)
```

CHAPTER 5

Authentication

5.1 Client credentials

Client credentials tokens can be created without any user interaction and allow access to public spotify data. Here's an example of how to use it with the requests client

```
import os

from spoffy.io.requests import make_spotify

CLIENT_ID = os.environ["SPOTIFY_CLIENT_ID"]
CLIENT_SECRET = os.environ["SPOTIFY_CLIENT_SECRET"]


spotify = make_spotify(client_id=CLIENT_ID, client_secret=CLIENT_SECRET)

spotify.auth.authorize_client()

# You can now make requests and access public spotify data

artist = spotify.artists.artist("64mPnRMMeudAet0E62ypkx")
print(artist.name, "sucks!")
```

5.2 OAuth2 login

Spoffy implements the authorization code login flow. This is described in great detail in Spotify's Authorization guide

Here is a simple example using flask to implement this oauth flow in a web backend and print the user's all time top artist.

First you will need to create an app ID and secret at https://developer.spotify.com/dashboard/ and set your redirect URI to http://localhost:5000/callback (if running your app at a different port/hostname substitute accordingly).

```
import os

from flask import Flask, url_for, redirect, request
from spoffy.io.requests import make_spotify


app = Flask(__name__)

app.config["CLIENT_ID"] = os.environ["SPOTIFY_CLIENT_ID"]
app.config["CLIENT_SECRET"] = os.environ["SPOTIFY_CLIENT_SECRET"]
app.config["SCOPE"] = "user-top-read"


def get_spotify():
    return make_spotify(
        client_id=app.config["CLIENT_ID"],
        client_secret=app.config["CLIENT_SECRET"],
        scope=app.config["SCOPE"],
        redirect_uri=url_for("spotify_callback", _external=True),
    )


@app.route("/")
def authorize_spotify():
    spotify = get_spotify()
    return redirect(spotify.auth.get_authorize_url())


@app.route("/callback")
def spotify_callback():
    spotify = get_spotify()
    response_code = request.args["code"]
    spotify.auth.authorize_user(response_code)

    top_artists = spotify.library.top_artists(
        limit=1, time_range="long_term"
    ).items

    if not top_artists:
        return "<h1>You have no top artists, go listen to more music!</h1>"

    return "<h1>Your all time top artist is: {artist.name}</h1>".format(
        artist=top_artists[0]
    )
```

5.3 Refreshing an access token

When you get a token through the oauth login the access token expires after one hour. You also get a refresh token that can be used to reauthorize with the spotify api without the user having to log in again.

If you already have an authorized user on your Spoffy client, refreshing is simple:

```
spotify.auth.refresh_authorzation()
```

If you have a refresh token from somewhere else, simply pass a refresh token explicitly

```
s = make_spotify(client_id='my_client_id', client_secret='my_client_secret')
s.auth.refresh_authorization('my_refresh_token')
```

If you have a `UserToken` object from a previous session

```
s = make_spotify(
    client_id='my_client_id',
    client_secret='my_client_secret',
    token=my_token_object
)
s.auth.refresh_authorization()
```

CHAPTER 6

Install

```
pip install spoffy
```

Only python3.6 and up are supported

CHAPTER 7

API Reference

7.1 IO Implementations

7.2 Requests

7.3 AIOhttp

7.4 Spotify wrapper

Sync and async spotify API wrappers that wrap a client instance and expose the spotify api organized into individual modules

7.5 Client base

7.6 Exceptions

7.7 Sync API modules

7.8 Async API modules

7.9 Object model

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

S