
Webradio to Spotify

Release 2.0

Eric Daoud

Feb 15, 2021

CONTENTS:

- 1 Installation** **3**
- 2 API** **5**
- 3 Web Interface** **7**
- 4 Contribute** **9**
 - 4.1 Writing your own scraper 9
- 5 Indices and tables** **15**
- Python Module Index** **17**
- Index** **19**

As a big fan of Classic Rock living in France, I am very frustrated by the lack of good classic rock radio we have. I spent four months in St Louis, MO, and I had the chance to listen to [KSHE 95](#) every day, playing some of my favorite classic rock tunes. Unfortunately, I can't listen to this radio in France as they block it. Fortunately, their website shows the tune currently playing, as well as a few previous ones.

I decided to make myself an empty [Spotify playlist](#), and automatically add in the KSHE tracks. I also wanted to be able to add songs from other similar Classic Rock radio. So I built a reusable architecture that enables to register different web scrapers to get the radio playing history and add that into my playlist.

So far, I am able to get the songs from these radios:

- [KSHE95 \(St Louis, MO\)](#)
- [The Eagle 969 \(Sacramento, CA\)](#)
- [Q104.3 \(New York, NY\)](#)
- [102.9 MGK \(Philadelphia, PA\)](#)
- [95.5 KLOS \(Los Angeles, CA\)](#)

Feel free to ping me if you want to help!

INSTALLATION

To make it work, here's what to do.

First, you'll need to setup your Spotify developer account, and register an app. Find how [here](#). Once your app is created, you will have access to the following credentials:

- `client_id`
- `client_secret`
- `redirect_uri`

Find your `user_id` (your spotify username) and add these 4 credentials in a file called `.spotify-token.json`. You have a template here: [.spotify-token.json.dist](#). The app will need those to update tracks to your playlist.

Note: in this application, the redirect URI must be `http://localhost:9999/auth/callback`.

Once you're good, install the requirements in a virtual environment:

```
pip install virtualenv # if you don't have it already
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
```

The app uses an `sqlite` database to store all the songs it has downloaded so far. You have to initialize the database running this command: `make init-db`.

Here are the required steps to update your playlist with the latest songs from the KSHE radio:

- First, launch the server: `make start-api`. The app should now be running on `http://localhost:9999`.
- Then, open your browser and go to `http://localhost:9999/auth` to authenticate to Spotify.
- Finally, run `make update-playlist` to get the latest songs in your playlist.

The calls supported so far are:

- GET, localhost:9999/api: Check that the API is up
- GET, localhost:9999/auth: Authenticate for 3600 seconds
- GET, localhost:9999/api/update_playlist: Updates the playlist with the latest songs

WEB INTERFACE

Webradio to Spotify



Authenticate

Update Playlist

Inserted Songs

KSHEScraper



Blue Collar Man (Long Nights)
Styx - Pieces Of Eight



Comedown - Remastered
Bush - Sixteen Stone (Remastered)

EagleScraper



Time - 2011 Remastered Version
Pink Floyd - The Dark Side Of The Moon [Remastered]
(Remastered Version)



More Than a Feeling
Boston - Boston

Q1043Scrapper



18 and Life
Skid Row - Skid Row



Smoke On The Water - Remastered 2012
Deep Purple - Machine Head (Remastered)



Take The Money And Run
Steve Miller Band - Fly Like An Eagle



Dancing With Myself
Generation X - Essential

4.1 Writing your own scraper

If you want to add another website to populate the playlist, you can write a new scrapper in the `src.scraping` module.

Please follow these steps to do so:

- **Create a class whose names ends with `Scraper`, e.g: `YourScraper`** (although it should be explicit which website it crawls).
- Make that class inherit from `Scraper`
- **Call for `super ()` in its constructor, and pass it the URL of the webpage** to crawl and the `playlist_id` to upload the songs to. e.g:

```
player_url = 'https://radio.com/awesome-song-history'  
playlist_id = '3BCcE8T945z1MnfPWkFsfX'  
super(YourScraper, self).__init__(player_url, playlist_id)
```

- Override the `get_song_history` method, the first row should be:

```
soup, driver = self.scrap_webpage()
```

- Add your scraper in the `tests` folder:

```
class TestYourScraper(GenericScraperTest):  
    scraper = scraping>YourScraper()
```

- Add your scraper in the `src.playlist_updater.Updater` class:

```
self.scrapers = [  
    scraping.KSHEScraper(),  
    scraping.EagleScraper(),  
    scraping>YourScraper() # New scraper!  
]
```

- You're all set!

4.1.1 src

src package

Subpackages

src.application package

```
src.application.create_app()
```

Flask app factory that creates and configure the app.

Submodules

src.application.api module

```
src.application.api.index()  
src.application.api.update_playlist()
```

src.application.auth module

```
src.application.auth.auth()  
src.application.auth.callback()
```

src.application.web module

```
src.application.web.about()  
src.application.web.auth()  
src.application.web.index()  
src.application.web.sync()  
src.application.web.update()
```

src.application.wsgi module

Submodules

src.db module

```
class src.db.Song(**kwargs)  
    Bases: sqlalchemy.ext.declarative.api.Base  
    album_image  
    album_name  
    artist_name  
    created_at  
    duration_ms  
    explicit  
    playlist_id  
    popularity
```

scraper_name
song_name
spotify_uri
updated_at

src.playlist_updater module

class src.playlist_updater.**Updater**

Bases: object

add_songs_to_playlist (*spotify_songs, playlist_id*)

Add spotify songs to a playlist, using songs URI.

Parameters **spotify_songs** (*list (dict)*) – List of spotify songs

Returns Json response from the Spotify API

Return type json

filter_and_save_songs_to_db (*spotify_songs, scraper_name, playlist_id*)

Filter out songs that have already been added and add the remaining songs to the playlist.

Parameters

- **spotify_songs** (*list (dict)*) – List of spotify songs as dict
- **scraper_name** (*str*) – Scraper class name

Returns List of spotify songs that are not in the playlist yet

Return type list(dict)

scrap_and_update ()

Run the whole pipeline for every scraper:

- Scrap the concerned website and get their song history
- Search for the songs in Spotify
- Filter the songs already in playlist and save them to DB
- Add the filtered songs to the playlist

Returns Inserted songs

Return type list(dict)

search_songs_in_spotify (*radio_history*)

Retrieve songs informations from title and artist using Spotify Search API.

Parameters **radio_history** (*list (dict)*) – list of dict with title and artist as keys

Returns list of dict of spotify songs

Return type list(dict)

single_scraper_pipeline (*scraper*)

spotify_auth ()

Authenticates using Authorization Code Flow.

Returns URL to redirect to

Return type str

spotify_callback (*authorization_code*)

Function called by Spotify with access token in the request parameters.

Parameters **authorization_code** (*str*) – Authorization code

sync_db_with_existing_songs (*playlist_id*)

If the playlist already exist, look for songs in it and stores them in the local database so we don't add duplicates.

Parameters **playlist_id** (*str*) – Playlist ID

src.scraping module

Add new scrapers here. Please follow these steps to do so:

- Create a class whose names ends with *Scraper*, e.g: *YourScraper* (although it should be explicit which website it crawls).
- Make that class inherit from *Scraper*
- Call for *super()* in its constructor, and pass it the URL of the webpage to crawl and the *playlist_id* to upload the songs to. e.g:

```
player_url = 'https://radio.com/awesome-song-history'  
playlist_id = '3BCcE8T945z1MnfPWkFsfX'  
super(YourScraper, self).__init__(player_url, playlist_id)
```

- Override the *get_song_history* method, the first row should be:

```
soup, driver = self.scrap_webpage()
```

- Add your scraper in the [tests](./tests/test_scraping.py) folder:

```
class TestYourScraper(GenericScraperTest):  
    scraper = scraping>YourScraper()
```

- Add your scraper in the [src.playlist_updater.Updater](./src/playlist_updater.py) class:

```
self.scrapers = [  
    scraping.KSHEScraper(),  
    scraping.EagleScraper(),  
    scraping>YourScraper() # New scraper!  
]
```

- You're all set!

class src.scraping.**EagleScraper**

Bases: *src.scraping.Scraper*

get_song_history ()

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)


```
class src.scraping.KLOScrapper
```

```
Bases: src.scraping.Scraper
```

```
get_song_history ()
```

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)

```
class src.scraping.KSHEScraper
```

```
Bases: src.scraping.Scraper
```

```
get_song_history ()
```

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)

```
class src.scraping.Q1043Scrapper
```

```
Bases: src.scraping.Scraper
```

```
get_song_history ()
```

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)

```
class src.scraping.Scraper (player_url, playlist_id)
```

```
Bases: abc.ABC
```

```
abstract get_song_history ()
```

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)

```
scrap_webpage ()
```

Scrap the webpage. This function must be called first in the `get_song_history` implementation.

Returns soup and driver

Return type tuple

```
class src.scraping.WMGKScrapper
```

```
Bases: src.scraping.Scraper
```

```
get_song_history ()
```

Scrap the website and get its song history. This function must be overiden. Its implementation must return a list of dict with the following keys:

- title
- artist
- timestamp (can be null, it's not used so far)

src.spotify module

class `src.spotify.SpotifyApi`

Bases: `object`

add_tracks_to_playlist (*track_uris*, *playlist_id*)

Add spotify songs to playlist, using their URIs.

Parameters `track_uris` (*list*) – List of songs URIs.

Returns Reponse from the Spotify API

Return type `json`

check_playlist_exists (*playlist_id*)

get_track_uris_from_playlist (*playlist_id*)

Return the track URIs from the playlist

Returns the songs URIs

Return type `set`

search_track (*track_name*, *artist_name*)

Search for a track using the Spotify Search API.

Parameters

- **track_name** (*str*) – Track name
- **artist_name** (*str*) – Artist name

Returns Dict containing the song attributes

Return type `dict`

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

- src, 9
- src.application, 10
- src.application.api, 10
- src.application.auth, 10
- src.application.web, 10
- src.db, 10
- src.playlist_updater, 11
- src.scraping, 12
- src.spotify, 14

A

about () (in module *src.application.web*), 10
 add_songs_to_playlist ()
 (*src.playlist_updater.Updater* method), 11
 add_tracks_to_playlist ()
 (*src.spotify.SpotifyApi* method), 14
 album_image (*src.db.Song* attribute), 10
 album_name (*src.db.Song* attribute), 10
 artist_name (*src.db.Song* attribute), 10
 auth () (in module *src.application.auth*), 10
 auth () (in module *src.application.web*), 10

C

callback () (in module *src.application.auth*), 10
 check_playlist_exists () (*src.spotify.SpotifyApi*
 method), 14
 create_app () (in module *src.application*), 10
 created_at (*src.db.Song* attribute), 10

D

duration_ms (*src.db.Song* attribute), 10

E

EagleScraper (class in *src.scraping*), 12
 explicit (*src.db.Song* attribute), 10

F

filter_and_save_songs_to_db ()
 (*src.playlist_updater.Updater* method), 11

G

get_song_history () (*src.scraping.EagleScraper*
 method), 12
 get_song_history () (*src.scraping.KLOScraper*
 method), 13
 get_song_history () (*src.scraping.KSHEScraper*
 method), 13
 get_song_history () (*src.scraping.Q1043Scraper*
 method), 13
 get_song_history () (*src.scraping.Scraper*
 method), 13

get_song_history ()
 (*src.scraping.WMGKScraper* method),
 13
 get_track_uris_from_playlist ()
 (*src.spotify.SpotifyApi* method), 14

I

index () (in module *src.application.api*), 10
 index () (in module *src.application.web*), 10

K

KLOScraper (class in *src.scraping*), 12
 KSHEScraper (class in *src.scraping*), 13

P

playlist_id (*src.db.Song* attribute), 10
 popularity (*src.db.Song* attribute), 10

Q

Q1043Scraper (class in *src.scraping*), 13

S

scrap_and_update () (*src.playlist_updater.Updater*
 method), 11
 scrap_webpage () (*src.scraping.Scraper* method), 13
 Scraper (class in *src.scraping*), 13
 scraper_name (*src.db.Song* attribute), 10
 search_songs_in_spotify ()
 (*src.playlist_updater.Updater* method), 11
 search_track () (*src.spotify.SpotifyApi* method), 14
 single_scraper_pipeline ()
 (*src.playlist_updater.Updater* method), 11
 Song (class in *src.db*), 10
 song_name (*src.db.Song* attribute), 11
 spotify_auth () (*src.playlist_updater.Updater*
 method), 11
 spotify_callback () (*src.playlist_updater.Updater*
 method), 12
 spotify_uri (*src.db.Song* attribute), 11
 SpotifyApi (class in *src.spotify*), 14
 src (module), 9
 src.application (module), 10

`src.application.api` (*module*), 10
`src.application.auth` (*module*), 10
`src.application.web` (*module*), 10
`src.db` (*module*), 10
`src.playlist_updater` (*module*), 11
`src.scraping` (*module*), 12
`src.spotify` (*module*), 14
`sync()` (*in module src.application.web*), 10
`sync_db_with_existing_songs()`
 (*src.playlist_updater.Updater method*), 12

U

`update()` (*in module src.application.web*), 10
`update_playlist()` (*in module src.application.api*),
 10
`updated_at` (*src.db.Song attribute*), 11
`Updater` (*class in src.playlist_updater*), 11

W

`WMGKScraper` (*class in src.scraping*), 13