Interview test report for Deezer

Nicolas Vo

1. Goal

Given a dataset, set up a web page which allows for visualizing the most used tags for each artists.

2. Dataset

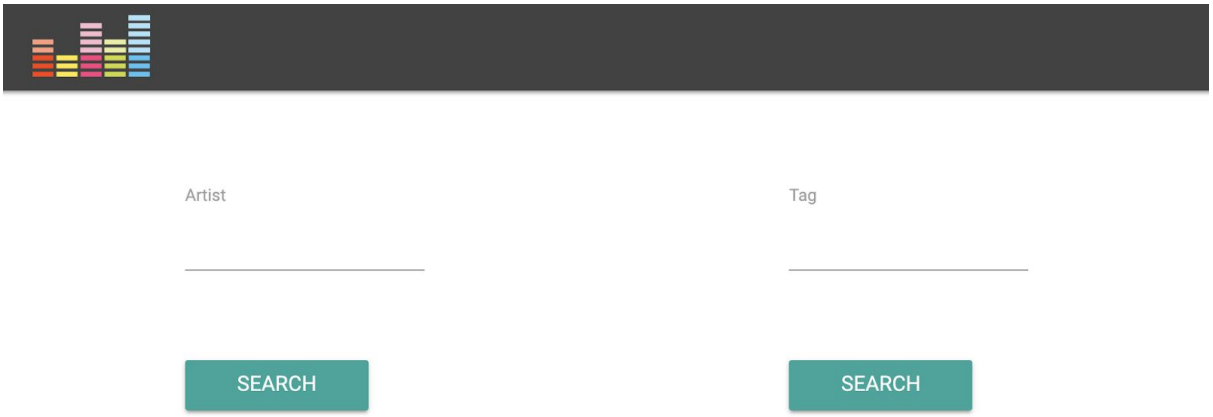
Last.fm dataset of artists, tags and the tags given to artists by users. Also given are the user's friends and the times at which the users tagged the artists.

3. Result

GitHub repository: https://github.com/nicolasvo/interview_deezer

3.1. Front

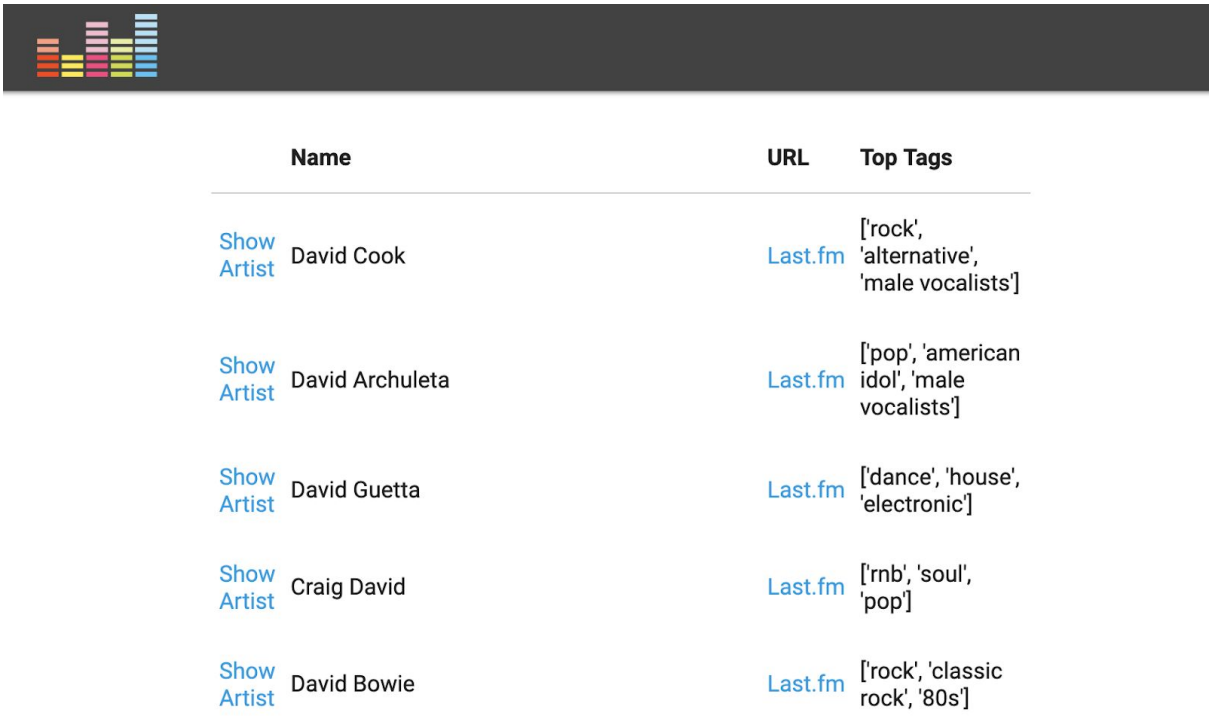
The web application produced lets a user search for either an artist or a tag.



Homepage

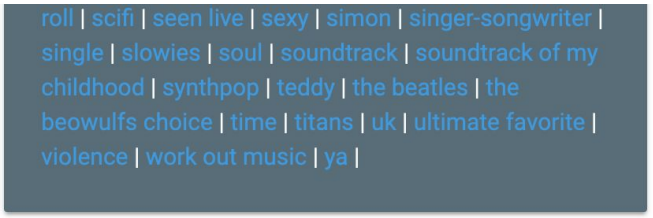
3.1.1. Artist

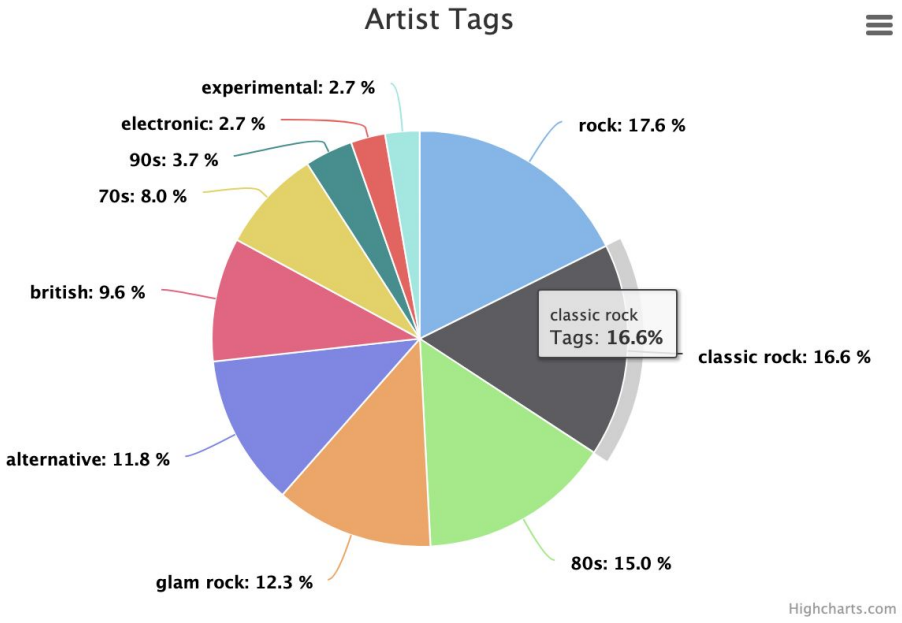
1. Enter an artist name
2. A list of artists are displayed (redirection link, name, last.fm link, top three tags)
3. On the artist page, all the tags of the artist are displayed in alphabetical order and are clickable (redirection to the tag page). A pie chart shows the top ten tags of the artist. Each slice is clickable and redirects to the wanted tag.



	Name	URL	Top Tags
Show Artist	David Cook	Last.fm	['rock', 'alternative', 'male vocalists']
Show Artist	David Archuleta	Last.fm	['pop', 'american idol', 'male vocalists']
Show Artist	David Guetta	Last.fm	['dance', 'house', 'electronic']
Show Artist	Craig David	Last.fm	['rnb', 'soul', 'pop']
Show Artist	David Bowie	Last.fm	['rock', 'classic rock', '80s']

List of artists with the name “david”





Pie chart of top 10 tags for David Bowie

3.1.2. Tag

1. Enter a tag name
2. A list of matching tags are displayed with the top three artists for each tag and a redirection to a specific tag
3. On the tag page are displayed the list of artists matching the tag in question.

Improvements:

- allow searching for both artist name and tag and only display results satisfying both conditions
- redirection link for each tag on the page with the list of matching artists
- redirection link for each artist on the page with the list of matching tags
- on the page of a specific tag, the list of artists displayed doesn’t include the top three tags of each artist because it takes time to get these tags and I did not want to show it again
- order results alphabetically

- a lot of portions of code are not DRY and in contrary quite repetitive. For some other parts, the code tries too much to be DRY for not much gain.

3.2. Back

3.2.1. Data importation

I used pandas dataframes and the pymongo API for the importation of the .dat files. The tag data required a latin-1 encoding. I converted the timestamps to datetime values in case I had ideas for filtering artists or tags by time.

3.2.2. Framework

- database with MongoDB: it is NoSQL, does not require a predetermined relational structure, therefore it is quick to set up.
- backend with Flask: it is lightweight in comparison to Django and does the job for such quickly deployed web apps. I used WTForms for the *very few* forms and validation I needed.
- visualizations with HighChartsJS: I find them elegant, responsive and interactive. However, I believe they are not as popular as D3.js graphs.

Improvements:

- when importing the data, group some document attributes. For instance, for an artist create a field with its tags and their occurrences in order to reduce loading times when querying.

3.3. Deployment

- pipenv: for the local work, I believe it is a standard much more reliable than simply pip and virtualenv alone
- Docker: deployed the final web app with it in order to make sure it could run smoothly on any machine. I also deploy the MongoDB on a Docker container.
- docker-compose: I enjoy simple container orchestration in order to deploy stuff with a single command