Assignment

Objective: Find and compile a list of at least 100,000 music artists with their name, genre, profile picture, and location, store this information in a database, and build a search backend and interface that allows users to search for artists with auto-suggestions for highly matching names.

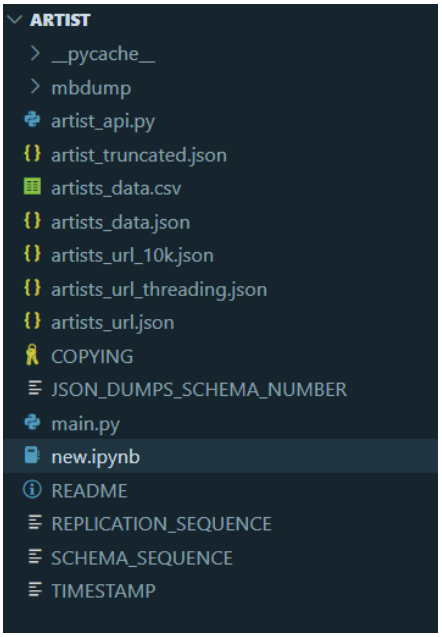
Dataset selected - MusicBrainz Database
The MusicBrainz Database is built on the PostgreSQL relational database engine and contains all of MusicBrainz' music metadata. This data includes information about artists, release groups, releases, recordings, works, and labels, as well as the many relationships between them. The database also contains a full history of all the changes that the MusicBrainz community has made to the data.

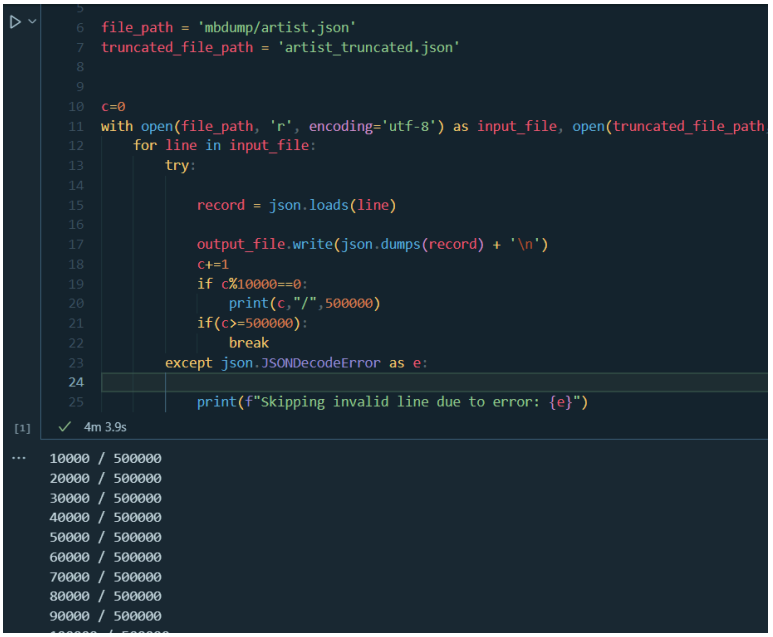
Dataset Downloaded - JSON Data dumps
Contained many relational tables including 'Artists'.

1) Extracted the tar.xz zip files

2) The original JSON file was very bulky - 12.8 GB so much that it couldn't be even opened as a normal .json file. Hence used Python for Data processing.

3) Removed unnecessary fields and compiled into ***artist_truncated.json***





4) Cleaned the data, Removed duplicates and any entries that do not have ‘Location’ and compiled into ***artists_data.json***

5) Used Beautiful Soup to get the profile picture of artists. So, wikimedia (wikipedia) too has a massive database of artists that has a very specific and structured way of storing profile pictures of artists.
`"https://en.wikipedia.org/wiki/{artist_name.replace(' ', '_')}"`

6) Hence, querying over the artists names and making API requests by replacing the url with the artist name would give the profile picture. In some cases however, there was no profile picture found.

7) A major constraint would be **rate limiting**, as too many API requests would lead to IP blocking. Hence I used batch processing. Created **110 batches** where each batch would process 1000 artists with a time.sleep of 0.5 seconds. Used **multi-threading** with reduced worker count

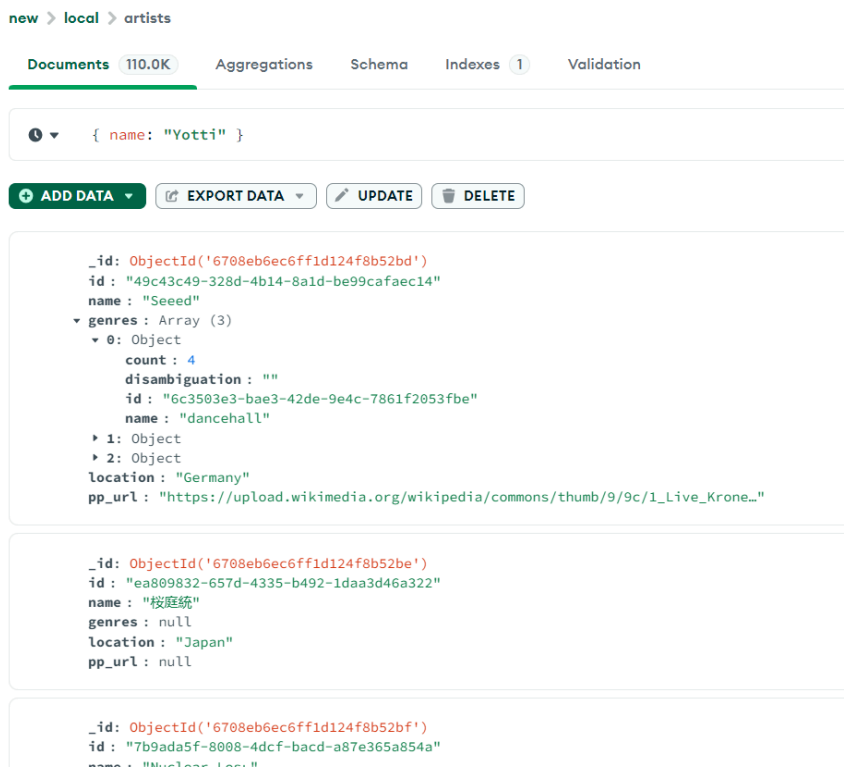
8) The output file at this stage artists_url_threading.json contains an additional field called **‘pp_url’** that contains the link to the profile picture. The final dataset has **1,10,000 artists**

9) Finally, I used MongoDB as it is very convenient to work with React + Node with MongoDB especially with .json files

```
Processing Batch 89/110: 100%|| 1000/1000 [00:11<00:00, 90.60it/s]
Processing Batch 90/110: 100%|| 1000/1000 [00:00<00:00, 2897.85it/s]
Processing Batch 91/110: 100%|| 1000/1000 [00:00<00:00, 4161.68it/s]
Processing Batch 92/110: 100%|| 1000/1000 [01:03<00:00, 15.70it/s]
Processing Batch 93/110: 100%|| 1000/1000 [00:48<00:00, 20.51it/s]
Processing Batch 94/110: 100%|| 1000/1000 [00:50<00:00, 19.96it/s]
Processing Batch 95/110: 100%|| 1000/1000 [00:43<00:00, 22.81it/s]
Processing Batch 96/110: 100%|| 1000/1000 [00:55<00:00, 17.98it/s]
Processing Batch 97/110: 100%|| 1000/1000 [00:47<00:00, 20.90it/s]
Processing Batch 98/110: 100%|| 1000/1000 [00:32<00:00, 30.93it/s]
Processing Batch 99/110: 100%|| 1000/1000 [00:36<00:00, 27.03it/s]
Processing Batch 100/110: 100%|| 1000/1000 [00:40<00:00, 24.39it/s]
Processing Batch 101/110: 100%|| 1000/1000 [00:36<00:00, 27.28it/s]
Processing Batch 102/110: 100%|| 1000/1000 [00:40<00:00, 24.92it/s]
Processing Batch 103/110: 100%|| 1000/1000 [00:35<00:00, 28.44it/s]
Processing Batch 104/110: 100%|| 1000/1000 [00:38<00:00, 25.92it/s]
Processing Batch 105/110: 100%|| 1000/1000 [00:45<00:00, 22.01it/s]
Processing Batch 106/110: 100%|| 1000/1000 [00:43<00:00, 23.10it/s]
Processing Batch 107/110: 100%|| 1000/1000 [00:43<00:00, 22.95it/s]
Processing Batch 108/110: 100%|| 1000/1000 [01:00<00:00, 16.48it/s]
Processing Batch 109/110: 100%|| 1000/1000 [00:39<00:00, 25.40it/s]
Processing Batch 110/110: 100%|| 1000/1000 [00:51<00:00, 19.50it/s]
Updated dataset saved as 'artists_url_threading.json'
```


```
{
        "id": "49c43c49-328d-4b14-8a1d-be99cafaec14",
        "name": "Seeed",
        "genres": [
```

```
            {
                "count": 4,
                "disambiguation": "",
                "id": "6c3503e3-bae3-42de-9e4c-7861f2053fbe",
                "name": "dancehall"
            },
            {
                "name": "hip hop",
                "id": "52faa157-6bad-4d86-a0ab-d4dec7d2513c",
                "disambiguation": "",
                "count": 2
            },
            {
                "name": "reggae",
                "id": "02b2f720-d06e-42ce-85c2-1ecd4191ffcb",
                "disambiguation": "",
                "count": 4
            }
        ],
        "location": "Germany",
        "pp_url":
"https://upload.wikimedia.org/wikipedia/commons/thumb/9/9c/1_Live_Krone
_2013_Seeed_2.jpg/260px-1_Live_Krone_2013_Seeed_2.jpg"
    },
```

Further using MERN Stack I implemented the Search Task.

Displaying artists:

Primary goal was to connect the backend with database and fire queries that would search for the artist name in the json file. The artist details would be displayed.

Recommendations:

Secondly, I used the prefix tree (Trie DS) approach to give suggestions for search. A prefix tree is generated at the time of search.
For example -

While searching for ‘Coldplay’

- Initially we would have a tree with 37 nodes (numbers from 0-9 and alphabets from A-Z)
- Now instead of parsing the entire database we would directly move to a node that contains ‘C’.
- When the user enters C, that particular branch is selected eliminating a major portion of the dataset making it very efficient.
- Further the next alphabet is entered i.e. ‘o’. So the second level tree having the branch containing ‘o’ is selected.
- In the similar I have allowed the algorithm to move up to 4 alphabets (as it is sufficient)

So when C is entered, all the words most related to C are suggested, which is rather vague But when Co is entered, there is a great chance that the word we want would already appear

Overcoming drawback of Trie DS

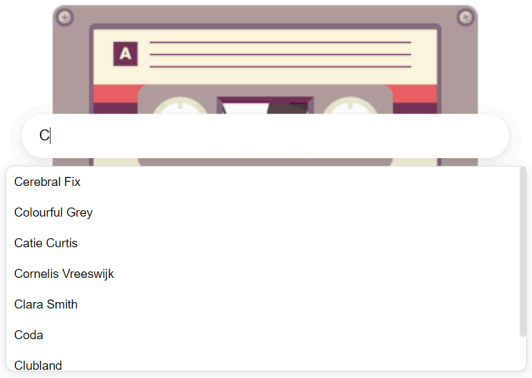
One major drawback of the prefix tree would be dependence on the preceding word. If I write Boldplay instead of Coldplay, most ideally I should be able to see Coldplay by suggestion but with the prefix tree I won’t be able to. Hence we can use the Fuzzy logic by Fuse.js

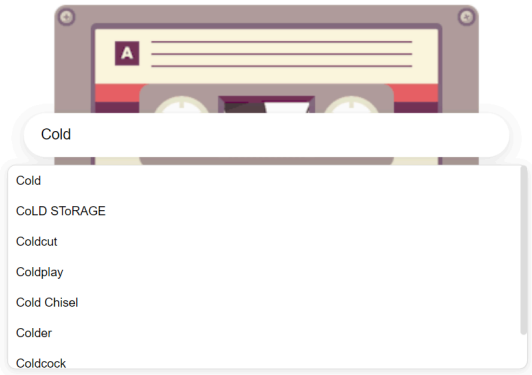
Fuse.js is a lightweight, JavaScript-based fuzzy-search library that helps perform fast and efficient searches, especially when working with a large amount of text data. Unlike standard search algorithms, which rely on exact matches, Fuse.js enables you to perform "fuzzy" searches—meaning it can find approximate matches even when the search input is misspelled or incomplete.

One key method Fuse.js uses is "Levenshtein distance," which is a metric for the minimum number of edits needed to turn one string into another. If you type "Boldplay," Fuse.js will calculate the distance between this and other artists in the dataset, determining that "Coldplay" is the closest.

Abbreviations

Simply convert all the words into abbreviations and store them along with the Trie. Add those to fuzzy logic as well





Power T

Power Tool
Powerstation
Dixie Power Trio
Power Man
Power Course
Power Symphony
Power Salad

Caldplay

Coldplay
Radical Playaz
Mandalay
Calla
Airplay
Calloway
Callahan's

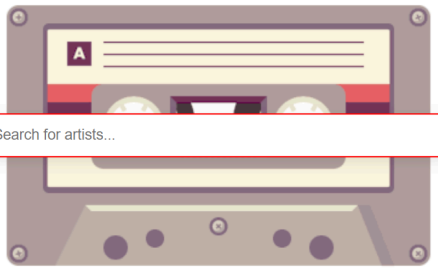
Boldplay

Coldplay
Fourplay
Morplay
Foreplay
Bolzplatz Heroes
Floorplay
Global Players

Boldplay

Artist Details

Name: Coldplay
Location: United Kingdom
Genres: alternative rock, art pop, britpop, dream pop, piano rock, pop, pop rock, post-britpop, rock



Search for artists...

MJ

No Image

Name: MJ Cole
Location: United Kingdom
Genres: uk garage

No Image

Name: MJP
Location: N/A
Genres: N/A

No Image

Name: MJG
Location: United States

MJ

Margaret Johnson

Michael Johnson

Merrill Jenson

Michael Jackson

MJ Cole

MJP

MJG