WebIR: A lyrics search engine

Aurélien VU NGOC ▪2020280219

Presenting ILYrics

A minimal lyrics search engine

- Music industry is thriving
- Everyday sees new song titles coming out
- Lot of streaming services, but **only few that display lyrics**

Existing industry products

- Google
- Genius
- Shazam
- ...

Demo

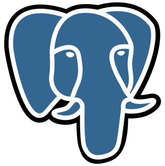
Outline

1. **Introduction & Demo**
2. Components
3. Functionalities
4. Data & Index
5. Query
6. Improvements
7. Conclusion

Outline

1. Introduction & Demo
2. **Components**
3. Functionalities
4. Data & Index
5. Query
6. Improvements
7. Conclusion

Components



Components

Why use these technologies?

Django - powerful tool to create web applications running Python

AWS PostgreSQL RDS - efficient, scalable & easy-to-use online relational database service. Django comes with fully integrated connector to PostgreSQL database.

AWS ElasticSearch Service - again, efficient, scalable & easy-to-use ElasticSearch service

Heroku - enables quick and in-the-cloud deployment of applications

Outline

1. Introduction & Demo
2. Components
3. **Functionalities**
4. Data & Index
5. Query
6. Improvements
7. Conclusion

Functionalities

User side

- **Highlights** query matches
- **Top result, Song results, Lyric matches**
- Can understand queries, even with **typo**
- Is **mobile friendly**

Developer side

- Automatically **refresh index** with new songs
- Fully integrated **maintenance tasks** (e.g. force duplicates search in index, bulk add data, synchronize database & elasticsearch

Outline

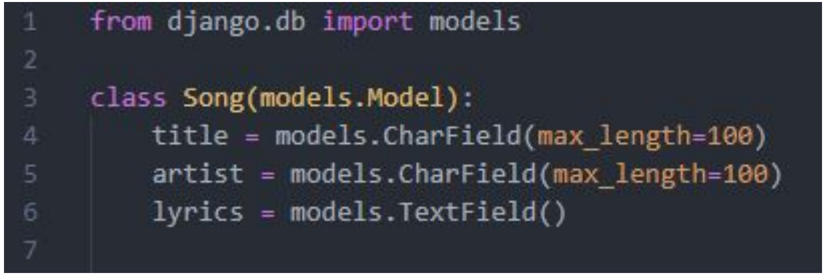
1. Introduction & Demo
2. Components
3. Functionalities
4. **Data & Index**
5. Query
6. Improvements
7. Conclusion

Data & Index

Data Sources

- Online **datasets** (kaggle, spotify billboard, …)
- **Scraping** on lyrics websites (genius.com, lyrics.com, azlyrics.com, …)

Data Structure



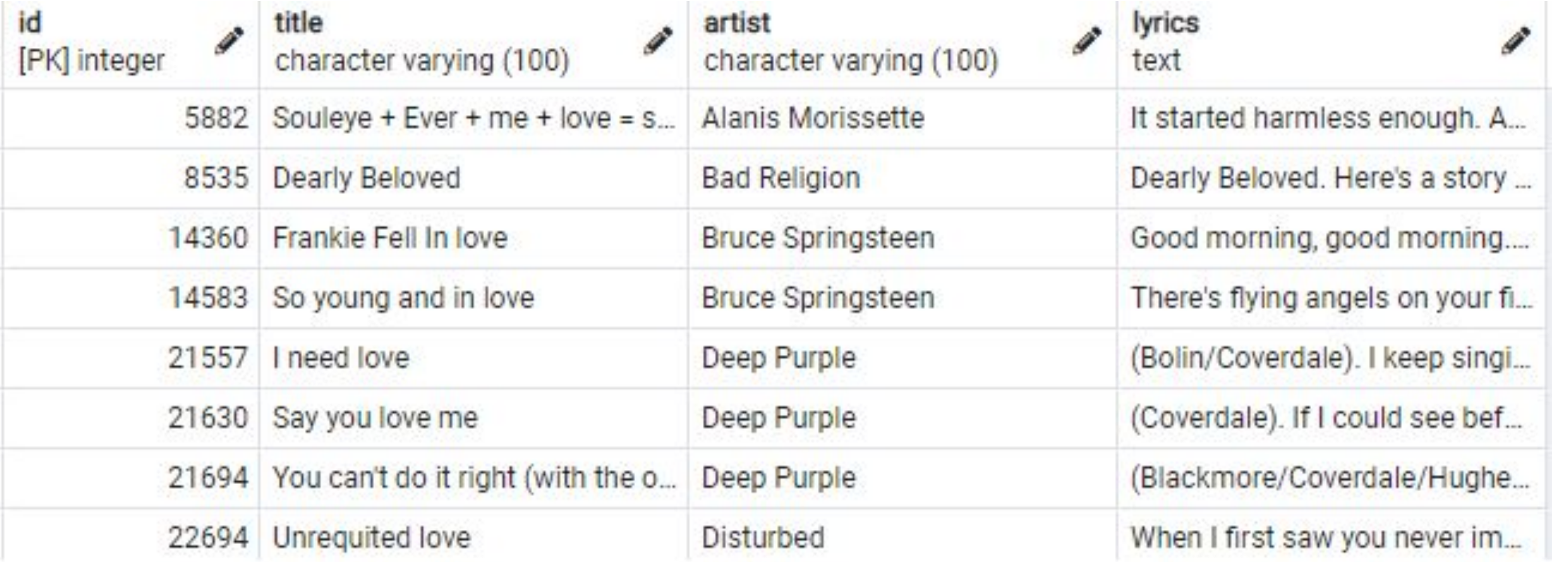
```python
from django.db import models

class Song(models.Model):
    title = models.CharField(max_length=100)
    artist = models.CharField(max_length=100)
    lyrics = models.TextField()

```

Data & Index

```
SELECT * FROM public."searchApp_song" where title like '%love%'
```



id [PK] integer	title character varying (100)	artist character varying (100)	lyrics text
5882	Souleye + Ever + me + love = s...	Alanis Morissette	It started harmless enough. A...
8535	Dearly Beloved	Bad Religion	Dearly Beloved. Here's a story ...
14360	Frankie Fell In love	Bruce Springsteen	Good morning, good morning....
14583	So young and in love	Bruce Springsteen	There's flying angels on your fi...
21557	I need love	Deep Purple	(Bolin/Coverdale). I keep singi...
21630	Say you love me	Deep Purple	(Coverdale). If I could see bef...
21694	You can't do it right (with the o...	Deep Purple	(Blackmore/Coverdale/Hughe...
22694	Unrequited love	Disturbed	When I first saw you never im...

Data & Index

Update index

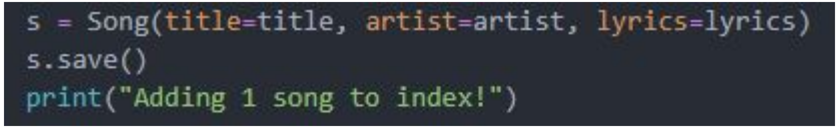
- **Live scraping** to update index

- **Procedure:**

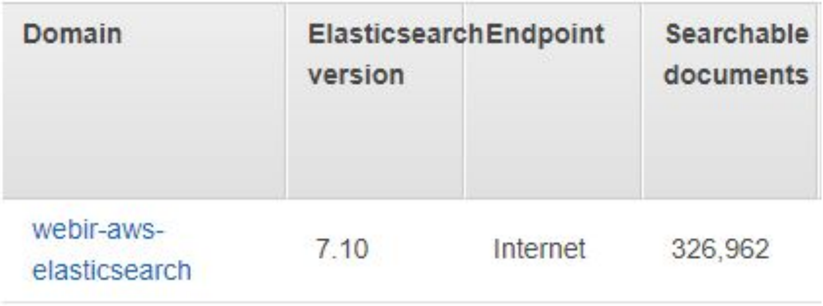
```
Foreach query:
        Update index with songs related to this query
EndFor
```

How?

- Using django models and elasticsearch



```
s = Song(title=title, artist=artist, lyrics=lyrics)
s.save()
print("Adding 1 song to index!")
```



Domain	Elasticsearch version	Endpoint	Searchable documents
webir-aws-elasticsearch	7.10	Internet	326,962

Outline

1. Introduction & Demo
2. Components
3. Functionalities
4. Data & Index
5. **Query**
6. Improvements
7. Conclusion

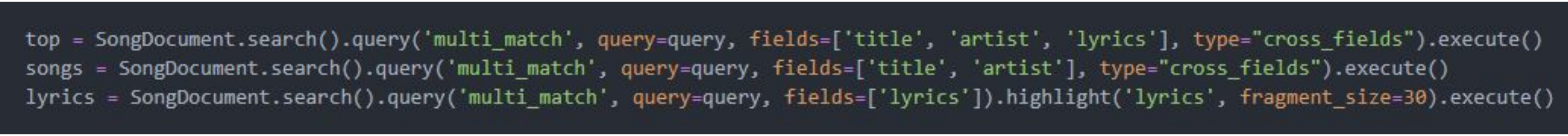
Query

Parsing

- Strip and split
- Then parse to be elasticsearch-friendly

3 levels of queries

- **Top result**: look into all fields
- **Song results**: look into “title”+”artist”
- **Lyric matches**: look into “lyrics” only



```
top = SongDocument.search().query('multi_match', query=query, fields=['title', 'artist', 'lyrics'], type="cross_fields").execute()
songs = SongDocument.search().query('multi_match', query=query, fields=['title', 'artist'], type="cross_fields").execute()
lyrics = SongDocument.search().query('multi_match', query=query, fields=['lyrics']).highlight('lyrics', fragment_size=30).execute()
```

Query

Ranking

- ElasticSearch runs Apache Lucene, thus uses **Lucene's Practical Scoring Function**

- Combination of **Boolean Model**, **TF/IDF** and **Vector Space model**

Scoring function (bm25-like)

- *queryNorm(q)* is the query normalization factor
- *coord(q,d)* is the coordination factor
- *t.getBoost()* is the boost that has been applied to the query
- *norm(t,d)* is the field-length normalization

$$score(q,d) = queryNorm(q) \cdot coord(q,d) \cdot \sum_{t \in q} \left[tf(t \in d) \cdot idf(d)^2 \cdot t.getBoost() \cdot norm(t,d) \right]$$

https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html

Outline

1. Introduction & Demo
2. Components
3. Functionalities
4. Data & Index
5. Query
6. **Improvements**
7. Conclusion

Improvements

Possible improvements

- **Scoring function**: experiment with new ranking methods, or elaborate a custom one

- **Query understanding**: better understand queries, using e.g. PLMs

- **Parsing**: improve parsing pipeline

- **Scale**: gather more data & re-structure tables with artist models, album models, …

- **Security**: fully integrate IAM connection (or more) on AWS

Conclusion

ILYrics is an efficient yet minimal search engine with plenty of room for improvements.

ILYrics in figures

- 326,962 indexed songs and growing
- 66 commits on GitHub
- 8000+ lines of code

https://github.com/smeelock/ilyrics

Thank you!