`lyriqs.io`

1. Application description

`lyriqs.io` is an application for looking up song lyrics and analysing their sentiment with natural language processing. The user is able to search for songs and select one from a list of results. The lyrics are then fetched for the song of the user's choice and displayed as plain text along with a chart showing their sentiment. Also, a word cloud is built from the lyrics and placed alongside the text. The song information and lyrics are queried from the `Musixmatch` API and then subject to a sentiment analysis performed by the `text-processing.com` API which is built on top of the Natural Language Toolkit (NLTK). The sentiment results are displayed in a chart created with `D3.js`. The word cloud of the lyrics is created using the `QuickChart` API.

2. Use cases

1. As a user interested in music, I want to be able to search for songs and find their lyrics.
2. As a user, would like to find out whether the mood of my favourite song is more positive or negative.
3. As a user, I want to get an overview of the most common words in the lyrics of my favourite song.

3. API description

Musixmatch API: API to search and retrieve song information and lyrics. Endpoints used are `GET track.search` to search for songs and `GET track.lyrics.get` to retrieve the lyrics. Authentication happens via an API key retrieved after setting up an account and 2k calls per day are free. However, apparently only 30% of the lyrics are accessible with a free plan, but for the sake of the exercise this should be fine (I also struggled to find better APIs for song lyrics).

text-processing.com API: Simple JSON over HTTP web service for text mining and natural language processing. It is currently free and open for public use without authentication, but limited to 1k calls per IP per day. The only endpoint used is `POST sentiment`.

QuickChart's Word Cloud API: Free charting API to create word or tag clouds without restrictions and no need for authentication. The endpoint used is `POST wordcloud`.

D3.js: This charting library is used to draw the donut chart for the sentiment analysis results.

S3: AWS' object store used for the page counter.

4. Application architecture

The application consists of two sub-projects: `lyriqs-server` and `lyriqs-client`. Both sub-projects contain a `Dockerfile` to account for separation of concerns, encapsulation, abstraction and several other principles of software architecture. The project itself includes a `Docker Compose YAML` file which allows

to run multiple containers side by side without interfering with each other. The idea is that the whole project can be run by executing `docker compose up` from the root project.

The **server** is a simple `node.js` backend using the web application framework `express.js`. It is responsible for sending out REST requests to the various APIs and processing the responses. The mashing of the search results is also performed by the server. After the user selects a specific song from Musixmatch, its ID will be used to send requests to fetch the lyrics from the same API. The lyrics are then used as the input to the other two APIs to perform the sentiment analysis and to retrieve the word cloud representation.
The following results of the server requests are presented to the client:

- song search results: song name and artist (JSON)
- plain text lyrics (string)
- sentiment distribution as confidence levels for positive, neutral, negative (JSON)
- word cloud representation (base64-encoded string)

The **client** is an Angular application and responsible for displaying the results of the server requests. The other main task of the client is the processing of user input like the query term for the song search as well as the choice of song from the results. The client will also wrap the positive/negative results of the sentiment analysis in a donut chart created with the `D3.js` library. The neutral value will be displayed next to the chart to provide additional information. The base64-encoded string of the word cloud will be converted to an image in the client. This happens easily by directly feeding the string into an HTML image tag.

Note: I will try to write the whole project (server and client) in TypeScript as I simply like the language more.