Set.lystr

Antero Duarte
40211946@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09183)

Abstract

Set.lystr is a music catalogue exploration/playlist creation tool geared towards musicians who play cover songs. Set.lystr allows its users to search and compare songs and build the ultimate playlist.

Keywords – Advanced, Web, Technologies, Python, Flask, MusicBrainz, AcousticBrainz

1 Introduction

Set.lystr This web application aims to be more than a music catalogue website. By providing users with acoustic metrics, such as BPM (beats per minute), a song's musical key and others, a user can meticulously plan their their setlist/playlist for the best flow.

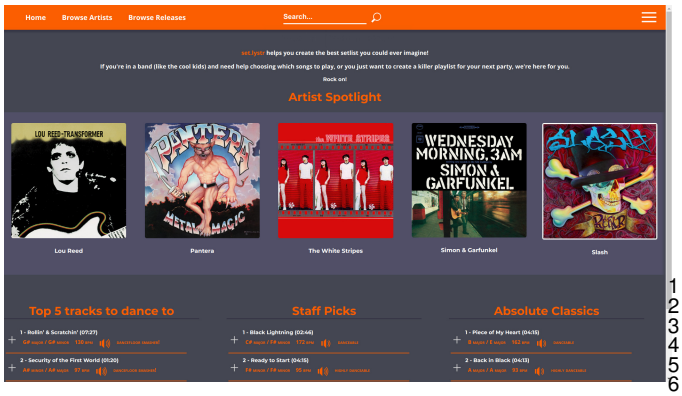

Figure 1: **ImageTitle** - Home page

Apart from said metrics, the website also provides music exploration capabilities, by listing every release created by a selection of well known artists, and every track in each release. If a user is not sure of what they are looking for, they can browse artists and releases. If a user knows exactly what they want, they can search and get a list of artists, releases and tracks that match their search terms. Set.lystr uses MusicBrainz[**?**] as a data source, and links to acoustic data from AcousticBrainz [**?**]. Once a user has found a song they would like to include in their playlist, they can add it to their personal setlist, which builds up while informing the user of the selected songs metrics.

2 Design

This web app was designed in line with modern standards and best practices. All the data gathered is available in the public domain, and the only library used was iconocss[**?**], which is a pure css icon library.

2.1 Data gathering

Data was gathered by collating a list of artist names from a list of the top 100 pop/rock bands on IMDb[**?**] and my personal spotify playlists. The total number of artists after merging the lists, removing duplicates, removing artist names such as "Various artists" and removing guilty pleasures was 192. Using Node.js[**?**], I "scraped" the MusicBrainz API to lookup artists based on their name from the list, as seen on listing 3 in the appendices. After hthis I downloaded every release (listing **??**)

2.2 Maths

Embedding Maths is Latex's bread and butter

2.3 Code Listing

You can load segments of code from a file, or embed them directly.

```
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
    std::cin.get();
    return 0;
}
```

Listing 1: Hello World! in c++

```
print "Hello World!"
```

Listing 2: Hello World! in python script

2.4 PseudoCode

3 Conclusion

4 Appendices

4.1 Data gathering code

for $i = 0$ **to** 100 **do**
    print_number = true;
    if *i is divisible by 3* **then**
        print "Fizz";
        print_number = false;
    end
    if *i is divisible by 5* **then**
        print "Buzz";
        print_number = false;
    end
    if *print_number* **then**
        print i;
    end
    print a newline;
end

Algorithm 1: FizzBuzz

```
const mb = require('musicbrainz')
async function matchToMusicBrainz() {
    let done = 0
    // bands => [String] => all the band names gathered
    const promises = bands.map(async (b) => {
        try {
            const [first] = await searchArtists(b)
            done += 1
            console.error(`${done}/${bands.length}`)
            return {
                id: first.id,
                name: first.name,
                country: first.country,
                lifespan: [first.lifeSpan.begin, first.lifeSpan.end]
            }
        } catch (e) {
            console.error(e)
        }

    })
    const results = await Promise.all(promises)
    results.forEach(r =>{
        console.log(`"${r.id}","${r.name}","${r.country}","${r.lifespan[0]} ${r.lifespan[1]}"`)
    })
}
// Immediately Invoked Function Expression, because async/await can't be used at top level
(async ()=>{
 await matchToMusicBrainz()
})()

function searchArtists(query, filter, force) {
    return new Promise((resolve, reject) => {
        mb.searchArtists(query, filter, force, (err, data) => {
            if (err) return reject(err)
            return resolve(data)
        })
    })
}
```

Listing 3: Artist name to MusicBrainzID lookup

```
const mb = require('musicbrainz')
const fcsv = require('fast-csv')
const axios = require('axios')

async function getMusicBrainzAlbums() {
  let done = 0;
  fcsv.fromPath('./csv/musicbrainzID.csv', {
    headers: ['id', 'name', 'country', 'lifespan']
  }).on('data', async ({
    id
  }) => {
    const artist = await lookupArtist(id, ['release-groups'])
    const albums = artist.releaseGroups.filter(r => r.type == 'Album')
    const promises = albums.map(async (a) => {
      let mainrelease,front,back,stage;
      try {
        stage = 'relgroup'
        const {
          data
        } = await axios.get(`https://coverartarchive.org/release-group/${a.id}/`);
        stage = 'coverart-start';
        mainrelease = (data.release || '').split('/').pop();
        stage = 'coverart-rel';
        front = data.images.find(i => i.front).image
        stage = 'coverart-front';
        back = data.images.find(i => i.back).image
        stage = 'coverart-end';
      } catch (e) {
        console.error(`failed: ${artist.name} - ${a.title} (${stage}) => ${a.id}`)
      } finally {
        switch(stage){
          case 'coverart-end':
            // artist id, relgroup id, relgroup name, first release date, main release, front cover, back cover
            console.log(`"${id}","${a.id}","${a.title}","${a.firstReleaseDate || ''}","${mainrelease}","${front || ''}","${back || ''}"`)
            break
          case 'coverart-front':
            // artist id, relgroup id, relgroup name, first release date, main release, front cover
            console.log(`"${id}","${a.id}","${a.title}","${a.firstReleaseDate || ''}","${mainrelease}","${front || ''}",""`)
            break
          case 'coverart-rel':
            // artist id, relgroup id, relgroup name, first release date, main release
            console.log(`"${id}","${a.id}","${a.title}","${a.firstReleaseDate || ''}","${mainrelease}","",""`)
            break
          default:
            // artist id, relgroup id, relgroup name, first release date
            console.log(`"${id}","${a.id}","${a.title}","${a.firstReleaseDate || ''}","","","",""`)
        }
        done += 1
        console.error(`${done}/192`)
        return Promise.resolve()
      }
    })
    return Promise.all(promises)
  })
}

function lookupArtist(id, links) {
  return new Promise((resolve, reject) => {
    mb.lookupArtist(id, links, (err, data) => {
      if (err) return reject(err)
      return resolve(data)
    })
  })
}
```

Listing 4: Release fetching from Musicbrainz. Includes cover art url fetching

This step gathered errors in lookups and classified them at different stages:

1. coverart-end - has all data
2. coverart-front - has main release and front cover
3. coverart-rel - has main release
4. relgroup - has relgroup only

For the relgroup stage, a lot of the releases are just bootlegs, or live albums, so they can safely be ignored For the coverart-rel stage, a lot of them do have cover images, they're just not in the expected format. I picked 2/3 of the more important ones and manually collected that data. For the coverart-front stage, we're only missing a back cover, which is not that important For the coverart-end stage, we have all data, which is ideal.

Before removing the relgroups, had 2065, after removing 206 relgroups, ended up with 1859

The release lookup listed all songs in a record. At the end of the lookup, 18 did not match, so a second pass was made to download the rest. Code:

```
const fcsv = require('fast-csv')
const mb = require('musicbrainz')

async function getMusicBrainzSongs() {
  return new Promise((resolve, reject) => {
    let done = 0;
    fcsv.fromPath('./csv/albums-full.csv', {
      headers: ["artist", "relgroup", "relgroup-name", "releasedate", "mainrel", "frontcover", "backcover"]
    }).on('data', async ({
      mainrel
    }) => {
      const release = await lookupRelease(mainrel, ['recordings'])
      release.mediums.forEach(m => {
        const format = m.format && m.format['#'] ? m.format['#'] : m.position;
        m.tracks.forEach(t => {
          try {
            // release id, medium, track id, track name, position, length
            console.log(`"${mainrel}","${format.replace(/"/g,'')}","${t.recording.id}","${t.recording.title}","${t.position}","${t.length}"`)
          } catch (e) {
            console.error(`failed: ${t.recording.id} => ${e.message}`)
          }
        });
      });
      done += 1;
      console.error(`${done}/1858`)
    }).on('end', () => {
      resolve(true)
    })
  })
}


(async () => {
  await getMusicBrainzSongs()
})()

function lookupRelease(id, links) {
  return new Promise((resolve, reject) => {
    mb.lookupRelease(id, links, (err, data) => {
      if (err) return reject(err)
      return resolve(data)
    })
  })
}
```

Listing 5: Track fetching from musicbrainz

After this, the songs were matched to records in acousticbrainz, a website which holds acoustic data about these songs. The following fields were recorded:

beats per minute, average loudness, chord change rate, chord key, chord scale, song key, song scale, key strength

with the following code:

```
const fcsv = require('fast-csv')
const axios = require('axios')

async function getAcousticBrainz() {
  let done = 0;
  const promises = [];
  let row = 0;
  fcsv.fromPath('./csv/tracks.csv', {
    headers: ["release", "medium", "track", "name", "position", "length"]
  }).on('data', async ({
    track
  }) => {
    const elem = (async (currRow) => {
      return new Promise(async (resolve, reject) => {
        try {
          await new Promise((r) => {
            setTimeout(r(), (row % 10) * 100)
          })
          const {
            data
          } = await axios.get(`https://acousticbrainz.org/api/v1/${track}/low-level`)
          const bpm = (data && data.rhythm && data.rhythm.bpm) || "0"
          const loud = (data && data.lowlevel && data.lowlevel.average_loudness) || "0"
          const chordchange = (data && data.tonal && data.tonal.chords_changes_rate) || "0"
          const chordkey = (data && data.tonal && data.tonal.chords_key) || ""
          const chordscale = (data && data.tonal && data.tonal.chords_scale) || ""
          const keykey = (data && data.tonal && data.tonal.key_key) || ""
          const keyscale = (data && data.tonal && data.tonal.key_scale) || ""
          const keystr = (data && data.tonal && data.tonal.key_strength) || "0"
          console.log(`"${track}","${bpm}","${loud}","${chordkey} ${chordscale}","${chordchange}","${keykey} ${keyscale}","${keystr}"`)
          done += 1
          console.error(`${done}/23519`)
          resolve(true)
        } catch (e) {
          console.error(`failed (${row}): ${track} => ${e.message}`)
        }
      })
    })(row)
    promises.push(elem)
    row++
  }).on('end', () => {
    return Promise.all(promises)
  })
}


(async () => {
  await getAcousticBrainz()
})()
```

Listing 6: Matching individual tracks in musicbrainz with entries in acousticbrainz

Out of the original 23519 songs gathered from the previous step, 3587 were not found in acousticbrainz, because acousticbrainz does not have an entry for 100% of the songs in musicbrainz. After a while, because the volume of calls was so big, acousticbrainz started responding with 50x errors, but by recording these errors and doing a 2nd and a 3rd pass, I managed to download the rest of the songs, with the total count being 19932.

Way into the server building, I realised I forgot to download a field that I wanted to have, danceability. So I partially re-wrote the code above to download the whole json file for the song's acoustics, so I could just go through the files if I realised I forgot something else.

```
const fcsv = require('fast-csv')
const axios = require('axios')

async function downloadAcousticBrainz() {
  let done = 0;
  const promises = [];
  let row = 0;
  fcsv.fromPath('./csv/fullbrainz.csv', {
    headers: ["release", "medium", "track", "name", "position", "length"]
  }).on('data', async ({ track }) => {
    const elem = (async (currRow) => {
      return new Promise(async (resolve, reject) => {
        try {
          const exists = await fileExists(`./acoustics/${track}.json`)

          if (!exists) {
            await new Promise((r) => {
              setTimeout(r(), (currRow % 10) * 100)
            })
            const {
              data
            } = await axios.get(`https://acousticbrainz.org/api/v1/${track}/low-level`)
            fs.writeFile(`./acoustics/${track}.json`, JSON.stringify(data),(err)=>{
              if(!err){
                done+= 1
                console.log(`${done}/19933`)
                resolve(true);
              }else{
                throw err
                resolve(true)
              }
```

```
            });
          }else{
            done+=1
            console.log(`skipping ${track}, done: ${done}/19933`)
          }
        } catch (e) {
          console.error(`failed (${currRow}): ${track} => ${e.message}`)
        }
      })
    })(row)
    promises.push(elem)
    row++
  }).on('end', async () => {
    promises.push(new Promise((r) => {
      setTimeout(() =>{
        console.log("waiting 60 secs for any unresolved promise to resolve")
        r();
      },  60000)
    }))
    return Promise.all(promises)
  })
}


(async () => {
  await downloadAcousticBrainz()
})()
```

Listing 7: Downloading the whole json file for each acousticbrainz entry

This code also allows for multiple passes to be executed without messing with logs, which means I could run this in an infinite loop, where I waited about 10 minutes to let the acousticbrainz servers come back after getting 50x errors.

Also while doing this, I realised that even after a few passes, I was only getting maximum 19048 files, which lead me to investigate. I realised that there are duplicate entries in the tracks.csv file, which made it seem like there are more songs than the 19048

After that I created a function that would go through all downloaded files and actually gather the metrics I was looking for:

```
const fcsv = require('fast-csv');
const axios = require('axios');
const { promisify } = require('util');
const fs = require('fs');
const fileExists = promisify(fs.exists);
const readdir = promisify(fs.readdir);

const readFile = promisify(fs.readFile)
async function parseAcousticBrainz() {
  const files = await readdir("./acoustics");
  // print the headers
  console.log('"id","bpm","loudness","chord_key","chord_change_rate","song_key","key_strength","danceability","dynamic_complexity"')
  files.forEach(async name => {
    try {
      const file = await readFile('./acoustics/' + name)
      // Remove the .json bit from the file name to get the ID back
      const id = name.substring(0, name.length - 5)
      const data = JSON.parse(file);
      const bpm = (data && data.rhythm && data.rhythm.bpm) || "0"
      const danceability = (data && data.rhythm && data.rhythm.danceability) || "0"
      const loud = (data && data.lowlevel && data.lowlevel.average_loudness) || "0"
      const dc = (data && data.lowlevel && data.lowlevel.dynamic_complexity) || "0"
      const chordchange = (data && data.tonal && data.tonal.chords_changes_rate) || "0"
      const chordkey = (data && data.tonal && data.tonal.chords_key) || ""
      const chordscale = (data && data.tonal && data.tonal.chords_scale) || ""
      const keykey = (data && data.tonal && data.tonal.key_key) || ""
      const keyscale = (data && data.tonal && data.tonal.key_scale) || ""
      const keystr = (data && data.tonal && data.tonal.key_strength) || "0"
      console.log(`"${id}","${bpm}","${loud}","${chordkey} ${chordscale}","${chordchange}","${keykey} ${keyscale}","${keystr}","${danceability}","${dc}"`)
    } catch (e) {
      console.error(`Failed: ${name} => ${e.message}`)
    }
  })
}


(async () => {
  await parseAcousticBrainz()
})()
```

Listing 8: Parsing the acousticbrainz files to turn the relevant metrics into csv format