Integrate Internet Archive in BrainzPlayer

Proposed mentors: *lucifer*, *monkey*
Languages/skills: Python/Flask, Typescript/React, Postgres
Estimated Project Length: 350 hours
Difficulty: medium
Expected Outcomes: A deployable component to play internet archive music

ListenBrainz has a number of music discovery features that use BrainzPlayer to facilitate track playback. BrainzPlayer (BP) is a custom React component in LB that uses multiple data sources to search and play a track. The 78 RPMs and Cylinder Recordings is a collection of digitised recordings from physical releases of the early 20th century. Each recording comes with audio streaming, and metadata web service. It can be used to retrieve metadata automatically and to embed a player in ListenBrainz using BrainzPlayer. A lot of similar music collections are hosted by the Internet Archive which can be made available for playing on ListenBrainz.

Personal Information

Name: Harij Sharma
Email Address: sharmaharij21@gmail.com
Github/IRC: huhridge
Location: Delhi, India
Time Zone: GMT+5:30
University: Delhi Technological University
Degree: Mathematics and Computing (Bachelor of Technology)
Year of Study: 2

Project Introduction

Currently ListenBrainz uses a few listening services to allow users to play tracks. These include YouTube, Spotify, Apple Music and Soundcloud. The initial idea of this project was to add Internet Archive as well to this list of providers.

Following discussions with the mentors and the community on the IRC, I have come to the conclusion that there is so much more that can be accomplished by adding IA to the list of providers. There is a lot of music that is on the Internet Archive that is not available on the streaming services, this can be highlighted in a way, and used in the MusicBrainz database and make it more complete.

My PRs: Here

Goals

- A component that searches the Internet Archive for the track and serves it up as an option to the BrainzPlayer component.
- A metadata cache and content resolver for Internet Archive that crawls a given collection and collects metadata to be used later in the MB database.

Implementation

The implementation is quite straightforward, just like the various players in BrainzPlayer, a react component similar to the YoutubePlayer, that searches the Internet Archive and makes it available for playing.

Looking into the Internet Archive API, particularly the search API (here), a simple component that makes a GET request and searches the IA seems to be the simplest and most reliable approach. This approach does not require any authorization tokens.

On searching a few tracks, it seems that a lot of "restricted" items are returned, this can be easily rectified in the module by tuning the search parameters.

The query proposed is:

```
title:(*Title*) AND creator:(*Artist*) AND mediatype:(audio) AND
 -access-restricted-item:(true)
```

The function proposed to be used for searching tracks on the Internet Archive is:

```
const fetchIAAudios = async (identifier?: string): Promise<Object> => {
  const response = await fetch(
    `https://archive.org/metadata/${identifier}/files`,
    {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    }
  );
  const files = await response.json();
  const audios = files.result.filter((file: { source: string; title: any }) => {
  // filters out all the unwanted files and returns only the mp3 file that can be used
    return file.source === "derivative" && Boolean(file.title);
  });
  return audios;
};

// returns an Internet Archive Item of the specific song, if found
const searchForIATrack = async (
  trackName?: string,
  artistName?: string,
  releaseName?: string
): Promise<Object | null> => {
  let query = trackName ?? "";
  if (releaseName) {
    query = `(${query}) OR (${releaseName})`;
  }
  if (artistName) {
    query += ` AND creator:${artistName}`;
  }
  query += " AND mediatype:(audio) AND -access-restricted-item:(true)";
// access-restricted-item ensures samples are not returned

  const response = await fetch(
    `https://archive.org/advancedsearch.php?q=${encodeURIComponent(
      query
    )}&fl[]=identifier&fl[]=creator&fl[]=title&sort[]=downloads+desc&output=json&rows=100`,
    {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    }
  );

  const responseBody = await response.json();
  if (
    responseBody.responseHeader.status !== 0 ||
    responseBody.response.numFound === 0
  ) {
    return null;
  }
  let ident = "";
  for (const creator of responseBody?.response.docs?.[0]?.creator) {
    if (distanceCompare(artistName, creator)) {
      ident = responseBody.response.docs[0].identifier;
    }
  }
```

```
  if (ident) {
    const audios = await fetchIAAudios(ident);
    const foundTrack = audios.filter((audio: { title: string }) => {
      return audio.title.includes(trackName);
    });
    return foundTrack[0];
  }
  return null;
};
```

To find correct matches, a `distanceCompare` function will be used, that will use the Jaccard distance algorithm to tokenise and compare the returned artist name with the original. This is done to avoid issues where, the artist name is "John Doe", and the returned one is "Doe, John" and similar cases. Thanks to *monkey* for pointing that out.

The above functions will search Internet Archive for the track, and return an Item that contains all the details of the track:

```
    {
      name: '08 - Malihini Mele - Dorothy Lamour - Dick McIntire And His Harmony.mp3',
      source: 'derivative',
      creator: 'Dorothy Lamour;Dick McIntire And His Harmony Hawaiians;R. Alex Anderson',
      title: 'Malihini Mele (restored)',
      track: '07',
      album: 'Malihini Mele',
      genre: 'Hawaiian',
      bitrate: '135',
      format: 'VBR MP3',
      original: '08 - Malihini Mele - Dorothy Lamour - Dick McIntire And His Harmony.flac',
      mtime: '1598655726',
      size: '2696626',
      md5: '29acbd8c181195aa247dd6dcbc466329',
      crc32: 'f208aec6',
      sha1: '3d4d63c957d0195be6895e1677d949b2a97678d1',
      length: '152.54',
      height: '562',
      width: '640',
      identifier: '78_a-song-of-old-hawaii_dorothy-lamour-dick-mcintire-and-his-harmony-..'
    }
```

This can be used in the function below:

```
  searchAndPlayTrack = async (listen: Listen | JSPFTrack): Promise<void> => {
    const trackName = getTrackName(listen);
    const artistName = getArtistName(listen);
    const releaseName = _get(listen, "track_metadata.release_name");
    const { handleError, handleWarning, onTrackNotFound } = this.props;
    if ((!trackName && !releaseName) || !artistName) {
// can't find a song if we don't atleast have either track/album and artist name
      handleWarning(
        "We are missing a track title, artist or album name to search on Internet Archive",
        "Not enough info to search on Internet Archive"
      );
      onTrackNotFound();
      return;
    }

    try {
      const IATrack = await searchForIATrack(
        trackName,
        artistName,
        releaseName
      );
```

```
      if (IATrack) {
        const IAUrl = `https://archive.org/download/${IATrack.identifier}/${IATrack.name}`;
        this.playStreamUrl(IAUrl);
        return;
      }
      onTrackNotFound();
    } catch (errorObject) {
      handleError(
        errorObject.message ?? errorObject,
        "Error searching on Internet Archive"
      );
    }
  };
```

The `playStreamUrl` method uses an `<audio>` element to enable simple playback and bind it to the BrainzPlayer.

Audio Component:

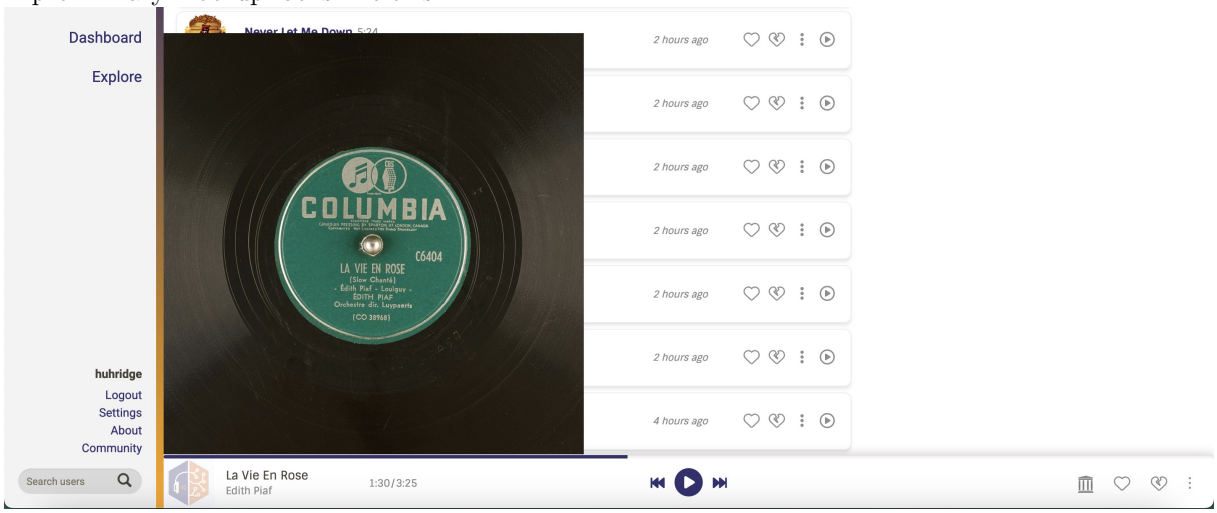
```
<div className={`internet-archive ${!show ? "hidden" : ""}`}>
        <audio
        id="internet-archive-player"
        src=""
        ref={this.IAPlayer}
        style={{display: "none"}}
        />
        <div>{this.getAlbumArt()}</div>
      </div>
```

The basic functionalities like play/pause, seekToMs and onTrackEnd can all be implemented fairly easily. The `<audio>` element has an `onended` event that can be used to trigger `onTrackEnd`.

```
  togglePlay = (): void => {
    if (!this.IAPlayer) {
      return;
    }
    if (this.IAPlayer.current!.paused) {
      this.IAPlayer.current!.play();
    } else {
      this.IAPlayer.current!.pause();
    }
  };

  seekToPositionMs = (msTimecode: number) => {
    if (!this.IAPlayer) {
      return;
    }
    this.IAPlayer.current!.currentTime = msTimecode;
  };
```

A preliminary mockup looks like this:



The second part of the project is building a metadata cacher, like the one that is already implemented for Spotify and Apple Music. Taking a look at the crawlers for those, they are seeded, i.e. populated, by incoming listens from those platforms and new releases served by the Spotify API, and charts from Apple Music.

```
    def get_items_from_listen(self, listen) -> list[JobItem]:
        album_id = listen["track_metadata"]["additional_info"].get("spotify_album_id")
        if album_id:
            return [JobItem(INCOMING_ALBUM_PRIORITY, album_id)]
        return []
```

This is not possible for Internet Archive, and the proposed alternative is building a crawler for a specific collection, like the 78 RPM collection. The proposed crawler will go through the whole collection and store the essential details from the metadata from each item.

Internet Archive provides us with a scrape endpoint in their API, and a function buried in the Search class of the python package exposes that endpoint.

```
from internetarchive import Search, ArchiveSession, get_item, get_files

s = ArchiveSession()
q = f'collection:{collection} AND creator:(*) AND mediatype:audio AND format:(MP3)
AND -access-restricted-item:(true)'
f=['creator','date','identifier','title','name','year','collection','language',
'format','description','external-identifier']
results = Search(s, q, fields=f)._scrape()
```

This method returns a generator that can be used to comb through the search results in a given collection.

Since IA is crowd sourced, there is no universal format the items follow. On going through a lot of search results, they can be divided into 2 categories, further divided:

- Items with a single track, that is a "single" album.
- Items with multiple tracks.

These two categories can be further divided into 2 categories on the basis of whether the files are tagged or not.

Depending in which category the item falls, the proposed implementation will process it differently. The basic premise is looping through the files list for the item, and after some basic filtering on tags, figuring out whether it's a single or an album and whether it is tagged.

```
# processes the returned search result and classifies it into a track or album
# and then processes it further
def process_item(item):
    files = get_files(identifier=item['identifier'], glob_pattern="*.mp3")
    files = list(files) #converting the generator returned to list
    item['files'] = files
```

```
    if len(files) == 1:
        process_album(item, 'single', tagged=hasattr(files[0], 'title'))
    else:
        for file in files:
            if hasattr(file, 'title'): #checks if the files are tagged and filters if so
                item['files'] = [file for file in files if hasattr(file, 'title')]
                process_album(item, tagged=True)
                return
        process_album(item)


def process_album(album, type='album', tagged=False):
    if tagged:
        tracks = []
        for i in range(len(album['files'])):
            track = album['files'][i]
            track_artists = [
                Artist(id='', name=artist, data={'name':artist})
                #TODO better split the artist name
                for artist in track.creator.split(";")
            ]
            tracks.append(Track(
                id=track.sha1,
                name=track.title,
                track_number=track.track,
                artists=track_artists,
                data=track
            ))

        artists = [
                Artist(id='', name=artist, data={'name':artist})
                #TODO better split the artist name
                for artist in album['creator'].split(";")
        ]

        date = None
        if hasattr(album, 'date'):
            date = album['date']

        return Album(
            id=album['identifier'],
            name=album['title'],
            type_=type,
            release_date=date,
            tracks=tracks,
            artists=artists,
            data=album
        )
```

It can be done similarly for the untagged type, just replacing the missing information with None.

As discussed on the IRC with *lucifer*, this is not a complete solution, and the content resolver will go through at least a dozen iterations, tested on a large set of data to gauge the correctness, which will be a great part of the project. Some of the improvements include better processing the creator name, as the format for it varies a lot in the search items. IA uses ";" to split the artists, but in a lot of cases, as the items are crowd-sourced, different characters have been used like ',' and '-', etc. A solution proposed by lucifer is using the Levenshtein distance algorithm to calculate confidence scores on the different methods of split and decide the best one. This approach shows premise and will be explored in the project.

Timeline

Pre-GSoC (April-May)

In this period, I plan to discuss and iron out all the final kinks in the project by chatting on the IRC, and looking over the community feedback and finalising the project.

Moving forward, I intend to continue contributing, especially to tickets related to this project.

For this period, my main goal would be to finalise all aspects, such as: UI, where to locate these functions, working out the search parameters. Basically familiarise myself with the implementation of the BrainzPlayer component.

Week 1
Looking into the Internet Archive API and the integrations made with Spotify and Apple Music, to figure the counterpart functions. Implement the first iteration of the searching and the metadata retrieving function.

Week 2
Testing the functions I have implemented in the first week, especially checking the searching function and dialing in the search parameters and making sure it works perfectly.

Week 3-4
Implementation of the new audio component that will play music from Internet Archive and integrate it with the existing BrainzPlayer component.

Week 5-6
Finalising the react component, checking all the integrations and basically making sure it works for all the different items (e.g. some items on the Internet Archive have multiple tracks, choosing the right one is crucial). Also finalising the front-end part. Also, I plan to use this week to catch up with some tasks that are not yet done.

Week 7-8
Implementation of the base metadata cache system such as the one used for Spotify and Apple Music.

Week 9-10
Testing the cache system and making sure the integration with the database is working. This also includes the different iterations and approaches that will be followed through and compared.

Week 11-12
Finalising the project and submitting the code for review. Discussing with mentors the final stages of the project.

I also plan to participate in the weekly Monday meetings on IRC to discuss the progress I've made in the week.

About Yourself

Hello, I am a second year student at DTU (formerly DCE), in the Mathematics and Computing Branch. In my first year, I studied DSA, full stack development, and was a software technician at UAS-DTU, the aerial robotics team of our college. I learned full systems integration, and worked with the ROS framework and implemented real time obstacle avoidance using cutting-edge kinodynamic algorithms.

I listen to a lot of music (Lisztomania), especially on Spotify and that is why I made some extensions for Spotify, using an open-source tool spicetify that lets you customise your client, to add some features that were requested by the community like the ability to list all your playlists with a specific songs, among many others.

Apart from that, I'm an avid reader (reading Dune at the moment), piano player and speedcuber!

- **Tell us about the computer(s) you have available for working on your SoC project!**
  I mainly work on MacBook Air (13-inch, M1, 2020) with 8 GB RAM. The Operating System - MacOS Sonoma 14.3.1.
- **When did you first start programming?**
  Back in 8th grade, I started to learn python and build little projects to just solve some problems I had. My first one was a simple renamer that would rename some songs I had downloaded according to their metadata. Another was a web scraper that I built to scrape metadata for some videos I had.

- **What type of music do you listen to?**
  I listen to literally everything, recently I've been on a hip-hop kick and have been listening to a lot of Frank Ocean, Tyler The Creator, and some more indie pop/rock artists such as Quadeca, Kevin Abstract and Rainbow Kitten Surprise.

  Some of my top songs at the moment are:
  Lovecrimes, by Frank Ocean: 6c40a19b-b174-4546-b71f-681dd9d05a65
  Texas Blue, by Quadeca and Kevin Abstract: 0f904db9-e668-4723-8a07-aa86ce036140
  Cold Love, by Rainbow Kitten Surprise: 41b9793f-492f-49fa-adeb-0de129f0177b
  Talk Down, by Dijon: 2a48cd74-1d0e-45ef-9be7-c3ae78312a7d
  Never Is a Promise, by Fiona Apple: dcf58245-319c-4f02-a3d6-ebd6f43ec289

  Come and follow me on LB !

- **What aspects of the project you're applying for (e.g., MusicBrainz, AcousticBrainz, etc.) interest you the most?**
  What I like most about ListenBrainz are the statistics and the recommendation system. I have gotten insights into my music taste and I have realised recently that I listen to a lot more hip-hop music than I thought I did. The troi recommendation system is also brilliant and has helped discover a lot of new music.

- **Have you ever used MusicBrainz to tag your files?**
  Yes! I used to download a lot of my music and Picard was the easiest way to tag them. Recently, I have used it again to tag unreleased Frank Ocean projects like nostalgia ULTRA, undocumented RARE, etc. and it has been incredibly useful.

- **Have you contributed to other Open Source projects? If so, which projects and can we see some of your code?**
  I have contributed to an Open Source project before, it's called spicetify. I mostly fixed a few bugs. I did use this project and developed my own extensions for Spotify, that you can check out here.

- **What sorts of programming projects have you done on your own time?**
  I have done a few projects, the spicetify extensions (here), a project on Real Time Obstacle Avoidance using ROS, which was used in ICUAS'23. Unfortunately, I am not allowed to share that code, my team did come 8th out of 40 other teams.

- **How much time do you have available, and how would you plan to use it?**
  In the summer, my university will not be ongoing, and I will devote all of my time into this project, i.e. at least 7-8 hours a day. At the moment, I am focussing at least 2 hours a day to understand the codebase better, and solve a few issues.