Proposal.md

GSoC 2020: Adding Statistics and Graphs for ListenBrainz Users and Community

Personal Information

- Name: Ishaan Shah
- IRC Nick: ishaanshah
- Email: ishaan.n.shah@gmail.com
- Github: https://github.com/ishaanshah
- Website: https://ishaanshah.github.io
- Time Zone: UTC +5:30

Project Overview

ListenBrainz now has a statistics infrastructure that collects and computes statistics from the listen data that has been stored in the database. Right now, the only information a user gets about their listening trends is a list of recent listens and top artists. This project aims to change this by displaying insightful graphs and statistics that would be more helpful to the user.

Graphs and Statistics which can be shown

We can classify the graphs and statistics to be shown in two different categories:

User Statistics

These graphs tell the user about their listening history and habits.

- **Listening Activity:** The number of listens submitted to ListenBrainz in the last week/month/year
- **Top Artist:** The top artists that the user has listened to
- **Top Releases:** The top releases that the user has listened to
- **Top Recordings:** The top recordings that the user has listened to
- **Daily Activity:** This graph shows the user activity during the day
- **Top Genres:** This graph shows the top 5 genres that user listens to
- **Artist Origins:** A map showing the locations of artists to which the user listens to
- **Mood Analysis:** Information such as *Danceability*, *BPM* and the general *Tone* of the songs that user listens to

Sitewide Statistics

These graphs tell about the sitewide trending artists, releases, and recordings in the ListenBrainz community. This data can also be used to calculate the popularity of the entities.

UI Mockups

This project will add three new views to serve the statistics that are being generated.

User statistics

UI Prototype
This view contains all the graphs and statistics that have been described in the User Statistics section.

User listen history

UI Prototype
This view shows a paginated list of the artists/recordings/releases that the user has listened to in a given time period.

Sitewide statistics

UI Prototype
This view shows the top 10 artists/recordings/releases that all ListenBrainz users are listening to. Moreover, the Listen Count shown on the homepage will be replaced by a graph showing the cumulative listens submitted to ListenBrainz over the last month.

Note: The mockups for the UI may change as per further discussions.

Implementation

Front End

ListenBrainz uses ReactJS for implementing UI components. I intend to use `nivo` - a React based charting library built using `d3.js` for rendering various visualizations. The reason to choose `nivo` as the charting library is -

- Has thorough and in-depth documentation
- Has a lot of customization options
- Has `typescript` definations, which help in devlopment considering that ListeBrainz is going to use `typescript` for `ReactJS` code in future
- Supports responsive components, which is essential in making the website mobile friendly

  The code used to build graphs for the mock UI can be found here.

Back End

Currently, listens are imported into Spark on the 8th and 22nd of every month. However, for the dynamic generation of graphs and statistics, the frequency of imports has to be increased. The listens should be imported every day at midnight, which means incremental data dumps have to be made every day.

Listen Activity

The listen activity shows the number of listens that a user has submitted over a period of time. It is a good measure of how active a user is and on which days is he most active.
Generating the data required is fairly easy. The psuedocode for generating the data for weekly listen activity is given below. The calculation of the data will be done only when the user visits the stats page
Psuedocode:

```
def get_listen_activity(week, user_name):
  message = {}
  df = get_listens(from_date=week.begin, to_date=week.end)
  df.createOrReplaceTempView("listens")
  for day in week:
    result = run_query("""SELECT *
                          FROM listens
                          WHERE user_name={user_name}
                          AND   timestamp>={utc(day.start)}
                          AND   timestamp<={utc(day.end)}
                        """)
    cnt = result.collect().count()
    message[day] = cnt
  return message
```

Top Artist/Recording/Release

The top artist/recording/release section shows the top 10 artist/recording/release that a user has listened to in the given period of time. Generating the data required for this is fairly easy. We first have to generate a `Dataframe` for the specified time period, then convert the `Dataframe` to a table and run the following `SQL` query on it. The calculation of the data will be done only when the user visits the stats page
SQL Query:

```
SELECT  artist_name
      , artist_msid
      , artist_mbid
      , COUNT(artist_name) as cnt
FROM    {table_name}
WHERE   user_name={user_name}
GROUP   BY  artist_name
          , artist_msid
          , artist_mbid
ORDER   BY  cnt
LIMIT   10
```

Similar queries can be made to get Top Recordings/Releases.

Daily Activity

The daily activity graph tells when in the whole day is a user most active. It reveales interesting information about a persons work habits and daily routine.
Calculating the data for the graph is moderately easy. The psuedocode for calculating this data is given below. As this data doesn't change much, it can be calculated once a week.
Pseudocode:

```
def get_daily_activity(week, user_name):
  df = get_listens(from_date=week.begin, to_date=week.end)
  df.createOrReplaceTempView("listens")
  data = [0 for i in range(1, 25)]
  for day in week:
    for hour in day:
      result = run_query("""SELECT *
                            FROM listens
                            WHERE user_name={user_name}
                            AND   timestamp>={utc(hour.start)}
                            AND   timestamp<={utc(hour.end)}
                          """)
      data[hour] += result.collect().count()
  data = [cnt/7 for cnt in data]

  return data
```

History

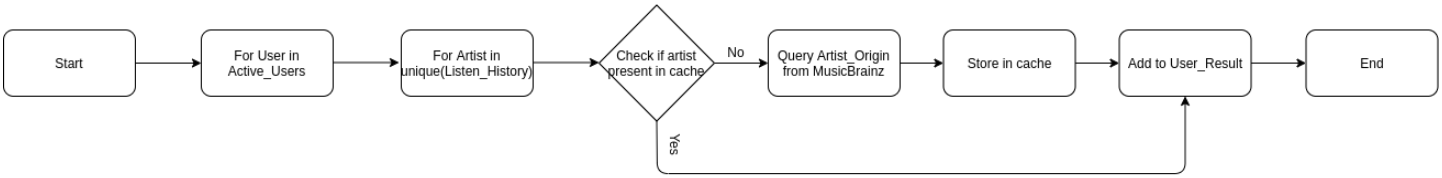
This section shows a paginated list of all the different entities that a user has listened to in a given time period.
A bar graph can be used to show this data. The `SQL` query to calculate this is similar to the `Top Artist` graph. The data will be calculated as and when the user requires it.

Sitewide Statistics

The sitewide statistics page shows the top artists/recordings/releases that the users are listening to in a week. These graphs reveal the current trending artists/releases/recordings.
A stream graph will be used to show which entity was most listened to in the past week. The `SQL` query to calculate the data required is similar to the `Top Artist` graph.

Artist Origin

The `Artist Origin` maps out all the artist's that a user has listened to. It shows how diverse a user's listening habits are.
This map is a bit difficult to implement as we have to query the MusicBrainz database to get the artist's origin. This data will be calculated weekly/biweekly, depending upon how fast this process is. A local cache can be created that maps various artists to their origin. This will make subsequent queries to get a particular artist's origin faster. The overall flow of the above process is shown in the figure -



Top Genres

The `Top Genres` chart shows the top 5 genres that user listened to.
A pie chart can be used to show this data. The data required to display this chart also has to be obtained from MusicBrainz database. Hence this data will be generated incrementally once per week. The overfall flow of the process is similar to the once shown in `Artist Origin`.

Mood Analysis

AcousticBrainz provides a lot of useful information such as **Danceability**, **BPM** and the general **Tone** of a recording. This can be used to provide insightful information about users' listening habits.
As the raw data provided by AB is hard relate to, this data will be shown relative to other users. For example, **Dancebility** of an user's songs is 20% more than average. As AB only supports `mbid` lookups for now, this project will calculate these statistics only for listens having a valid `recording_mbid`. Supporting all listens will be a stretch goal. These statistics will be calculated once in a week. Different statistics which can be shown are -

- BPM
- Dancebility
- Happiness
- Accousticness

Storing data for `Daily Activity, Sitewide Statistics`

As this data will be calculated only once per week, it has to be stored in ListenBrainz Server. This can be done by creating a table in `Postgress SQL` with the following schema -

Column	Type	Nullable
user_name	string	not nullable
data	jsonb	nullable
last_updated	timestamp	not nullable

Storing data for `Top Genres, Artist Origin and Mood Analysis`

The data for Artist Origin, Top Genres and Mood Analysis will be calculated incrementally. That is the data will be calculated for a week and then merged with previous data. Hence we have to store this data in HDFS. A new table with the following schema will have to be created for this.

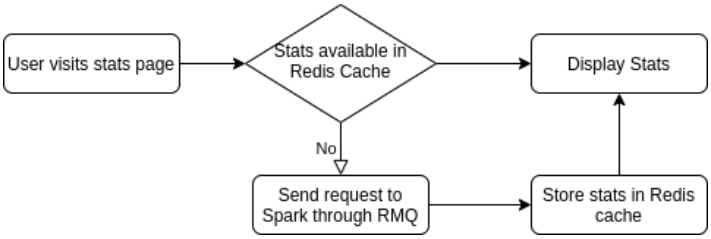
Column	Type	Nullable
user_name	string	not nullable
genre	map(string, integer)	nullable
artist_origin	map(string, integer)	nullable
mood	map(string, integer)	nullable
last_updated	long integer	not nullable

Redis cache

To improve the page loading time, we have to cache the results that we get from Spark in local memory. This can be done using Redis. An example request for getting data for `Listen Activity` will be,

```
"stats.user.listen_activity": {
  "name": "stats.user.listen_activity",
  "description": "Listening activity of user",
  "params": ["musicbrainz_id", "from", "to"]
}
```

We can **hash** the request JSON and use it as a key to store the result for the query and quickly retrieve it later if the same query is made. An entry stored in the cache will have a limited lifetime, after which it will be removed. This will ensure that the data gets updated after a suitable interval. The flow for the process is shown in the figure.



Timeline

Here is a more detailed week-by-week timeline of the 13 weeks GSoC coding period to keep me on track

Community Bonding Period

I will use this time to discuss implementation details with mentors. I will start configuring the ListenBrainz server to use `Typescript` and the Spark server to start generating statistics.

Week 1-2

Finalize and implement the UI for displaying user and sitewide statistics and write tests.

Week 3

Start working on the generation of user statistics.

Week 4 (Phase 1 evaluations here)

Complete user statistics.

Week 5

Write tests for user statistics generation. Refactor the code written before based upon feedback from mentors in evaluation.

Week 6

Implement `Redis Cache` and write tests.

Week 7

Implement sitewide statistics.

Week 8 (Phase 2 evaluations here)

Work on scripts to get information about the artist's origin and genre tags from MusicBrainz.

Week 9

Work on scripts to get information about the mood information from AcousticBrainz.

Week 10-11

Write backend code for `Top Genres`, `Artist Origin` and `Mood Analysis`.

Week 12-13

Buffer Period. Work on additional ideas.

Post GSoC / Additional Ideas

I would like to continue working with ListenBrainz after Summer of Code. This project aims at setting up basic architecture for generating statistics with Apache Spark. The addition of more statistics will be relatively easy.

`Mood Analysis` for listens not having `recording_mbid`

As mentioned in the proposal the project aims to implement `Mood Analysis` for listens having `recording_mbid` only. Support for all listens can be added later.

Entity Graphs

These graphs will show details about various entities like artists, recordings and releases, when did a user start listening to that entity.

Mainstream Meter

This measures how mainstream a users' musical choice is. This can be done by taking the popularity of an entity and number of listens for that entity into account.

About Me

I am a freshman at IIIT-H (International Institute of Information Technology, Hyderabad). I have been working with ListenBrainz since January and have learned quite a few things along the way. You can find the list of PRs that I have made over here.

Question: Tell us about the computer(s) you have available for working on your SoC project!

I have a HP laptop with an Intel i5 processor, and 8 GB RAM running Arch Linux. I also have a desktop computer with an Intel i7 processor, GTX 960 Graphics card, and 8 GB RAM running Arch Linux.

Question: When did you first start programming?

I have been programming since 10th grade. I started with C/C++ but now mostly code in Python and JavaScript.

Question: What type of music do you listen to?

I am a big fan of Coldplay. In addition to that, I also like listening to songs by Maroon 5, Lenka, and The Local Train.

Question: What aspects of ListenBrainz interest you the most?

The data collected by ListenBrainz is openly available and can be used to improve music technologies. Also the Open Source nature of ListenBrainz allow me to add features which are currently unavailable in closed source competitors.

Question: Have you ever used MusicBrainz to tag your files?

I have used MusicBrainz Picard to tag my music collection.

Question: Have you contributed to other Open Source projects? If so, which projects, and can we see some of your code?

ListenBrainz is the first open source organization that I have contributed to. However, I have done some other projects that can be seen on my Github Page.

Question: What sorts of programming projects have you done on your own time?

I wrote a bot that solved the Eight Puzzle as the final project for CS50. Recently I also worked on the platform used for Botomania, an onsite contest held at IIITH.

Question: How much time do you have available, and how would you plan to use it?

I plan to work for 35-45 hours per week as I would be on vacation during most of the coding period.

Question: Do you plan to have a job or study during the summer in conjunction with Summer of Code?

No