ListenBrainz Documentation

Release 0.1.0

MetaBrainz Foundation

Jan 20, 2022

API DOCUMENTATION

ListenBrainz is a project by the MetaBrainz foundation which allows you to publicly store a record of all of the songs that you listen to. Using this data, we provide statistics, recommendations, and a platform for you and other developers to explore this data.

If you want to use the ListenBrainz API to read or submit data, see the *API documentation*. You also may want to review the JSON documentation.

If you are interested in contributing to ListenBrainz as a developer, see the *Developer documentation.*

We also publish some maintainer documentation, which is used by the MetaBrainz team to run the ListenBrainz site.

CHAPTER

ONE

CONTENTS

1.1 ListenBrainz API

The ListenBrainz server supports the following end-points for submitting and fetching listens. All endpoints have this root URL for our current production site[1].

- **API Root URL**: `https://api.listenbrainz.org`
- **Web Root URL**: `https://listenbrainz.org`

Note: All ListenBrainz services are only available on **HTTPS**!

1.1.1 Reference

Core API Endpoints

`POST /1/submit-listens`

Submit listens to the server. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should also contain at least one listen in the payload.

Listens should be submitted for tracks when the user has listened to half the track or 4 minutes of the track, whichever is lower. If the user hasn't listened to 4 minutes or half the track, it doesn't fully count as a listen and should not be submitted.

For complete details on the format of the JSON to be POSTed to this endpoint, see *JSON Documentation*.

Request Headers

- Authorization – Token <user token>
- Content-Type – *application/json*

Status Codes

- 200 OK – listen(s) accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

[1] Provided for compatibility with other APIs, but we still recommend using `X-RateLimit-Reset-In` wherever possible

GET /1/validate-token

Check whether a User Token is a valid entry in the database.

In order to query this endpoint, send a GET request with the Authorization header set to the value `Token [the token value]`.

Note: This endpoint also checks for *token* argument in query params (example: /validate-token?token=token-to-check) if the Authorization header is missing for backward compatibility.

A JSON response, with the following format, will be returned.

- If the given token is valid:

```
{
    "code": 200,
    "message": "Token valid.",
    "valid": true,
    "user_name": "MusicBrainz ID of the user with the passed token"
}
```

- If the given token is invalid:

```
{
    "code": 200,
    "message": "Token invalid.",
    "valid": false,
}
```

Status Codes

- 200 OK – The user token is valid/invalid.
- 400 Bad Request – No token was sent to the endpoint.

POST /1/delete-listen

Delete a particular listen from a user's listen history. This checks for the correct authorization token and deletes the listen.

The format of the JSON to be POSTed to this endpoint is:

```
{
    "listened_at": 1,
    "recording_msid": "d23f4719-9212-49f0-ad08-ddbfbfc50d6f"
}
```

Request Headers

- Authorization – Token <user token>
- Content-Type – *application/json*

Status Codes

- 200 OK – listen deleted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*playlist_user_name*)**/playlists/collaborator**

Fetch playlist metadata in JSPF format without recordings for which a user is a collaborator. If a playlist is private, it will only be returned if the caller is authorized to edit that playlist.

Parameters

- **count** (int) – The number of playlists to return (for pagination). Default DEFAULT_NUMBER_OF_PLAYLISTS_PER_CALL
- **offset** (int) – The offset of into the list of playlists to return (for pagination)

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/user/(*playlist_user_name*)**/playlists/createdfor**

Fetch playlist metadata in JSPF format without recordings that have been created for the user. Createdfor playlists are all public, so no Authorization is needed for this call.

Parameters

- **count** (int) – The number of playlists to return (for pagination). Default DEFAULT_NUMBER_OF_PLAYLISTS_PER_CALL
- **offset** (int) – The offset of into the list of playlists to return (for pagination)

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/users/(*user_list*)**/recent-listens**

Fetch the most recent listens for a comma separated list of users. Take care to properly HTTP escape user names that contain commas!

Note: This is a bulk lookup endpoint. Hence, any non-existing users in the list will be simply ignored without raising any error.

Status Codes

- 200 OK – Fetched listens successfully.
- 400 Bad Request – Your user list was incomplete or otherwise invalid.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/similar-users**
Get list of users who have similar music tastes (based on their listen history) for a given user. Returns an array of dicts like these:

```
{
    "user_name": "hwnrwx",
    "similarity": 0.1938480256
}
```

Parameters

- **user_name** – the MusicBrainz ID of the user whose similar users are being requested.

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/listen-count**

Get the number of listens for a user `user_name`.

The returned listen count has an element 'payload' with only key: 'count' which unsurprisingly contains the listen count for the user.

Status Codes

- 200 OK – Yay, you have listen counts!
- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/playing-now**
Get the listen being played right now for user `user_name`.

This endpoint returns a JSON document with a single listen in the same format as the `/user/<user_name>/listens` endpoint, with one key difference, there will only be one listen returned at maximum and the listen will not contain a `listened_at` element.

The format for the JSON returned is defined in our *JSON Documentation*.

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/similar-to/**
other_user_name Get the similarity of the user and the other user, based on their listening history. Returns a single dict:

```
{
    "user_name": "other_user",
    "similarity": 0.1938480256
}
```

Parameters

- **user_name** – the MusicBrainz ID of the the one user
- **other_user_name** – the MusicBrainz ID of the other user whose similar users are

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*playlist_user_name*)**/playlists**

Fetch playlist metadata in JSPF format without recordings for the given user. If a user token is provided in the Authorization header, return private playlists as well as public playlists for that user.

Parameters

- **count** (int) – The number of playlists to return (for pagination). Default DEFAULT_NUMBER_OF_PLAYLISTS_PER_CALL
- **offset** (int) – The offset of into the list of playlists to return (for pagination)

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/listens**

Get listens for user user_name. The format for the JSON returned is defined in our *JSON Documentation*.

If none of the optional arguments are given, this endpoint will return the DEFAULT_ITEMS_PER_GET most recent listens. The optional max_ts and min_ts UNIX epoch timestamps control at which point in time to start returning listens. You may specify max_ts or min_ts, but not both in one call. Listens are always returned in descending timestamp order.

Parameters

- **max_ts** – If you specify a max_ts timestamp, listens with listened_at less than (but not including) this value will be returned.
- **min_ts** – If you specify a min_ts timestamp, listens with listened_at greater than (but not including) this value will be returned.
- **count** – Optional, number of listens to return. Default: DEFAULT_ITEMS_PER_GET . Max: MAX_ITEMS_PER_GET

Status Codes

- 200 OK – Yay, you have data!

- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/latest-import

Get the timestamp of the newest listen submitted by a user in previous imports to ListenBrainz.

In order to get the timestamp for a user, make a GET request to this endpoint. The data returned will be JSON of the following format:

```
{
    "musicbrainz_id": "the MusicBrainz ID of the user",
    "latest_import": "the timestamp of the newest listen submitted in previous␣
↪imports. Defaults to 0"
}
```

Query Parameters

- **user_name** (*str*) – the MusicBrainz ID of the user whose data is needed

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

POST /1/latest-import

Update the timestamp of the newest listen submitted by a user in an import to ListenBrainz.

In order to update the timestamp of a user, you'll have to provide a user token in the Authorization Header. User tokens can be found on https://listenbrainz.org/profile/.

The JSON that needs to be posted must contain a field named *ts* in the root with a valid unix timestamp. Example:

```
{
   "ts": 0
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – latest import timestamp updated
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Playlists API Endpoints

The playlists API allows for the creation and editing of lists of recordings

POST /1/playlist/create

Create a playlist. The playlist must be in JSPF format with MusicBrainz extensions, which is defined here: https://musicbrainz.org/doc/jspf . To create an empty playlist, you can send an empty playlist with only the title field filled out. If you would like to create a playlist populated with recordings, each of the track items in the playlist must have an identifier element that contains the MusicBrainz recording that includes the recording MBID.

When creating a playlist, only the playlist title and the track identifier elements will be used – all other elements in the posted JSPF wil be ignored.

If a created_for field is found and the user is not an approved playlist bot, then a 403 forbidden will be raised.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. The submitting user is not allowed to create playlists for other users.

Response Headers

- Content-Type – *application/json*

POST /1/playlist/ (*playlist_mbid*) **/item/delete**

To delete an item in a playlist, the POST data needs to specify the recording MBID and current index of the track to delete, and how many tracks from that position should be moved deleted. The format of the post data should look as follows:

```
{
    "index": 3,
    "count": 2
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. the requesting user was not allowed to carry out this operation.

Response Headers

- Content-Type – *application/json*

POST /1/playlist/ (*playlist_mbid*) **/item/move**

To move an item in a playlist, the POST data needs to specify the recording MBID and current index of the track to move (from), where to move it to (to) and how many tracks from that position should be moved (count). The format of the post data should look as follows:

```
{
    "mbid": "<mbid>",
    "from": 3,
    "to": 4,
    "count": 2
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – move operation succeeded
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. the requesting user was not allowed to carry out this operation.

Response Headers

- Content-Type – *application/json*

POST /1/playlist/ (*playlist_mbid*) **/item/add**

POST /1/playlist/ (*playlist_mbid*) **/item/add/**

int: *offset* Append recordings to an existing playlist by posting a playlist with one of more recordings in it. The playlist must be in JSPF format with MusicBrainz extensions, which is defined here: https://musicbrainz.org/doc/jspf .

If the offset is provided in the URL, then the recordings will be added at that offset, otherwise they will be added at the end of the playlist.

You may only add `MAX_RECORDINGS_PER_ADD` recordings in one call to this endpoint.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. the requesting user was not allowed to carry out this operation.

Response Headers

- Content-Type – *application/json*

POST /1/playlist/edit/ (*playlist_mbid*)

Edit the private/public status, name, description or list of collaborators for an exising playlist. The Authorization header must be set and correspond to the owner of the playlist otherwise a 403 error will be returned. All fields will be overwritten with new values.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. The subitting user is not allowed to edit playlists for other users.

Response Headers

- Content-Type – *application/json*

POST /1/playlist/ (*playlist_mbid*) **/delete**

Delete a playlist. POST body data does not need to contain anything.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist deleted.
- 401 Unauthorized – invalid authorization. See error message for details.
- 403 Forbidden – forbidden. the requesting user was not allowed to carry out this operation.
- 404 Not Found – Playlist not found

Response Headers

- Content-Type – *application/json*

POST /1/playlist/ (*playlist_mbid*) **/copy**

Copy a playlist – the new playlist will be given the name "Copy of <playlist_name>". POST body data does not need to contain anything.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – playlist copied.
- 401 Unauthorized – invalid authorization. See error message for details.
- 404 Not Found – Playlist not found

Response Headers

- Content-Type – *application/json*

GET /1/playlist/ (*playlist_mbid*)

Fetch the given playlist.

Parameters

- **playlist_mbid** (`str`) – The playlist mbid to fetch.
- **fetch_metadata** (`bool`) – Optional, pass value 'false' to skip lookup up recording metadata

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – Playlist not found
- 401 Unauthorized – Invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

Feedback API Endpoints

These API endpoints allow to submit and retrieve feedback for a user's recordings

POST /1/feedback/recording-feedback

Submit recording feedback (love/hate) to the server. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should contain only one feedback in the payload.

For complete details on the format of the JSON to be POSTed to this endpoint, see feedback-json-doc.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – feedback accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

GET /1/feedback/recording/(*recording_msid*)**/get-feedback**

Get feedback for recording with given `recording_msid`. The format for the JSON returned is defined in our feedback-json-doc.

Parameters

- **score** (`int`) – Optional, If 1 then returns the loved recordings, if -1 returns hated recordings.
- **count** (`int`) – Optional, number of feedback items to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`.
- **offset** (`int`) – Optional, number of feedback items to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 feedback will be skipped, defaults to 0.

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

GET /1/feedback/user/(*user_name*)**/get-feedback-for-recordings**

Get feedback given by user `user_name` for the list of recordings supplied. The format for the JSON returned is defined in our feedback-json-doc.

If the feedback for given recording MSID doesn't exist then a score 0 is returned for that recording.

Parameters

- **recordings** (str) – comma separated list of recording_msids for which feedback records are to be fetched.

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

GET /1/feedback/user/(*user_name*)**/get-feedback**

Get feedback given by user `user_name`. The format for the JSON returned is defined in our feedback-json-doc.

If the optional argument `score` is not given, this endpoint will return all the feedback submitted by the user. Otherwise filters the feedback to be returned by score.

Parameters

- **score** (int) – Optional, If 1 then returns the loved recordings, if -1 returns hated recordings.
- **count** (int) – Optional, number of feedback items to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`.
- **offset** (int) – Optional, number of feedback items to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 feedback will be skipped, defaults to 0.
- **metadata** (str) – Optional, 'true' or 'false' if this call should return the metadata for the feedback.

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

Recording Recommendation API Endpoints

ListenBrainz uses collaborative filtering to generate recording recommendations, which may be further processed to generate playlists for users. These api endpoints allow to fetch the raw collaborative filtered recording IDs.

GET /1/cf/recommendation/user/(*user_name*)**/recording**

Get recommendations sorted on rating and ratings for user `user_name`.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "last_updated": 1588494361,
        "type": "<artist_type>",
        "entity": "recording",
        "mbids": [
            {
                "recording_mbid": "526bd613-fddd-4bd6-9137-ab709ac74cab",
                "score": 9.345
            },
            {
                "recording_mbid": "a6081bc1-2a76-4984-b21f-38bc3dcca3a5",
                "score": 6.998
```


(continues on next page)

(continued from previous page)

```
            }
        ],
        "user_name": "unclejohn69",
        "count": 10,
        "total_mbid_count": 30,
        "offset": 10
    }
}
```

Note:

- This endpoint is experimental and probably will change in the future.
- <artist_type>: 'top' or 'similar'

Parameters

- **artist_type** (str) – Mandatory, artist type in ['top', 'similar']

  Ex. artist_type = top will fetch recommended recording mbids that belong to top artists listened to by the user.

  artist_type = similar will fetch recommended recording mbids that belong to artists similar to top artists listened to by the user.

- **count** (int) – Optional, number of recording mbids to return, Default: DEFAULT_ITEMS_PER_GET Max: MAX_ITEMS_PER_GET
- **offset** (int) – Optional, number of mbids to skip from the beginning, for pagination. Ex. An offset of 5 means the 5 mbids will be skipped, defaults to 0

Status Codes

- 200 OK – Successful query, you have data!
- 400 Bad Request – Bad request, check response['error'] for more details
- 404 Not Found – User not found.
- 204 No Content – Recommendations for the user haven't been generated, empty response will be returned

Recording Recommendation Feedback API Endpoints

ListenBrainz uses collaborative filtering to generate recording recommendations, which may be further processed to generate playlists for users. These api endpoints allow to submit and retrieve feedback for raw collaborative filtered recordings.

POST /1/recommendation/feedback/submit

Submit recommendation feedback. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should contain only one feedback in the payload.

A sample feedback may look like:

```
{
    "recording_mbid": "d23f4719-9212-49f0-ad08-ddbfbfc50d6f",
    "rating": "love"
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – feedback accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

POST /1/recommendation/feedback/delete

Delete feedback for a user. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should contain only one recording mbid in the payload. A sample feedback may look like:

```
{
    "recording_mbid": "d23f4719-9212-49f0-ad08-ddbfbfc50d6f",
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – feedback deleted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

GET /1/recommendation/feedback/user/(*user_name*)**/recordings**

Get feedback given by user `user_name` for the list of recordings supplied.

A sample response may look like:

```
{
    "feedback": [
        {
            "created": 1604033691,
            "rating": "bad_recommendation",
            "recording_mbid": "9ffabbe4-e078-4906-80a7-3a02b537e251"
        },
        {
            "created": 1604032934,
            "rating": "hate",
            "recording_mbid": "28111d2c-a80d-418f-8b77-6aba58abe3e7"
        }
    ],
    "user_name": "Vansika Pareek"
}
```

An empty response will be returned if the feedback for given recording MBID doesn't exist.

Parameters

- **mbids** (str) – comma separated list of recording_mbids for which feedback records are to be fetched.

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 404 Not Found – User not found.

Response Headers

- Content-Type – *application/json*

GET /1/recommendation/feedback/user/(*user_name*)

Get feedback given by user `user_name`.

A sample response may look like:

```
{
    "count": 1,
    "feedback": [
        {
            "created": "1345679998",
            "recording_mbid": "d23f4719-9212-49f0-ad08-ddbfbfc50d6f",
            "rating": "love"
        },
        "-- more feedback data here ---"
    ],
    "offset": 0,
    "total_count": 1,
    "user_name": "Vansika"
}
```

If the optional argument `rating` is not given, this endpoint will return all the feedback submitted by the user. Otherwise filters the feedback to be returned by rating.

Parameters

- **rating** (str) – Optional, refer to db/model/recommendation_feedback.py for allowed rating values.
- **count** (int) – Optional, number of feedback items to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`.
- **offset** (int) – Optional, number of feedback items to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 feedback will be skipped, defaults to 0.

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found.
- 400 Bad Request – Bad request, check `response['error']` for more details

Response Headers

- Content-Type – *application/json*

Statistics API Endpoints

ListenBrainz has a statistics infrastructure that collects and computes statistics from the listen data that has been stored in the database. The endpoints in this section offer a way to get this data programmatically.

GET /1/stats/sitewide/listening-activity

Get the listening activity for entire site. The listening activity shows the number of listens the user has submitted over a period of time.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "listening_activity": [
            {
                "from_ts": 1587945600,
                "listen_count": 26,
                "time_range": "Monday 27 April 2020",
                "to_ts": 1588031999
            },
            {
                "from_ts": 1588032000,
                "listen_count": 57,
                "time_range": "Tuesday 28 April 2020",
                "to_ts": 1588118399
            },
            {
                "from_ts": 1588118400,
                "listen_count": 33,
                "time_range": "Wednesday 29 April 2020",
                "to_ts": 1588204799
            }
        ],
        "to_ts": 1589155200,
        "range": "week"
    }
}
```


Note:

- This endpoint is currently in beta
- The example above shows the data for three days only, however we calculate the statistics for the current time range and the previous time range. For example for weekly statistics the data is calculated for the current as well as the past week.

Parameters

- **range** (str) – Optional, time interval for which statistics should be returned, possible values are *ALLOWED_STATISTICS_RANGE*, defaults to all_time

Status Codes

- 200 OK – Successful query, you have data!

- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details

Response Headers

- Content-Type – *application/json*

`GET /1/stats/sitewide/recordings`

Get sitewide top recordings.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "recordings": [
            {
                "artist_mbids": [],
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "recording_mbid": "0fe11cd3-0be4-467b-84fa-0bd524d45d74",
                "release_mbid": "",
                "release_name": "Delirium (Deluxe)",
                "track_name": "Love Me Like You Do - From \"Fifty Shades of Grey\"
↪"
            },
            {
                "artist_mbids": [],
                "artist_name": "The Fray",
                "listen_count": 23,
                "recording_mbid": "0008ab49-a6ad-40b5-aa90-9d2779265c22",
                "release_mbid": "",
                "release_name": "How to Save a Life",
                "track_name": "How to Save a Life"
            }
        ],
        "offset": 0,
        "count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- We only calculate the top 1000 all_time recordings
- `artist_mbids`, `artist_msid`, `release_name`, `release_mbid`, `release_msid`, `recording_mbid` and `recording_msid` are optional fields and may not be present in all the responses

Parameters

- **count** (int) – Optional, number of artists to return for each time range, Default: DEFAULT_ITEMS_PER_GET Max: MAX_ITEMS_PER_GET
- **offset** (int) – Optional, number of artists to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 artists will be skipped, defaults to 0
- **range** (str) – Optional, time interval for which statistics should be returned, possible values are *ALLOWED_STATISTICS_RANGE*, defaults to all_time

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check response['error'] for more details

Response Headers

- Content-Type – *application/json*

GET /1/stats/sitewide/releases

Get sitewide top releases.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "releases": [
            {
                "artist_mbids": [],
                "artist_name": "Coldplay",
                "listen_count": 26,
                "release_mbid": "",
                "release_name": "Live in Buenos Aires"
            },
            {
                "artist_mbids": [],
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "release_mbid": "",
                "release_name": "Delirium (Deluxe)"
            },
            {
                "artist_mbids": [],
                "artist_name": "The Fray",
                "listen_count": 25,
                "release_mbid": "",
                "release_name": "How to Save a Life"
            },
        ],
        "offset": 0,
        "count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note:

- This endpoint is currently in beta
- `artist_mbids`, `artist_msid`, `release_mbid` and `release_msid` are optional fields and may not be present in all the responses

Parameters

- **count** (`int`) – Optional, number of artists to return for each time range, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **offset** (`int`) – Optional, number of artists to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 artists will be skipped, defaults to 0
- **range** (`str`) – Optional, time interval for which statistics should be returned, possible values are *`ALLOWED_STATISTICS_RANGE`*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details

Response Headers

- Content-Type – *application/json*

GET /1/stats/sitewide/artists

Get sitewide top artists.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "artists": [
            {
                "artist_mbids": [],
                "artist_name": "Kanye West",
                "listen_count": 1305
            },
            {
                "artist_mbids": ["0b30341b-b59d-4979-8130-b66c0e475321"],
                "artist_name": "Lil Nas X",
                "listen_count": 1267
            }
        ],
        "offset": 0,
        "count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- `artist_mbids` and `artist_msid` are optional fields and may not be present in all the entries
- We only calculate the top 1000 artists for each time period.

Parameters

- **`count`** (`int`) – Optional, number of artists to return for each time range, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **`offset`** (`int`) – Optional, number of artists to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 artists will be skipped, defaults to 0
- **`range`** (`str`) – Optional, time interval for which statistics should be returned, possible values are *`ALLOWED_STATISTICS_RANGE`*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details

Response Headers

- Content-Type – *application/json*

`GET /1/stats/user/` (*user_name*) **`/listening-activity`**

Get the listening activity for user `user_name`. The listening activity shows the number of listens the user has submitted over a period of time.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "listening_activity": [
            {
                "from_ts": 1587945600,
                "listen_count": 26,
                "time_range": "Monday 27 April 2020",
                "to_ts": 1588031999
            },
            {
                "from_ts": 1588032000,
                "listen_count": 57,
                "time_range": "Tuesday 28 April 2020",
                "to_ts": 1588118399
            },
            {
                "from_ts": 1588118400,
                "listen_count": 33,
                "time_range": "Wednesday 29 April 2020",
                "to_ts": 1588204799
            },
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
```


(continues on next page)

(continued from previous page)

```
    }
}
```

Note:

- This endpoint is currently in beta
- The example above shows the data for three days only, however we calculate the statistics for the current time range and the previous time range. For example for weekly statistics the data is calculated for the current as well as the past week.
- For `all_time` listening activity statistics we only return the years which have more than zero listens.

Parameters

- **range** (`str`) – Optional, time interval for which statistics should be returned, possible values are *ALLOWED_STATISTICS_RANGE*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/daily-activity**

Get the daily activity for user `user_name`. The daily activity shows the number of listens submitted by the user for each hour of the day over a period of time. We assume that all listens are in UTC.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "daily_activity": {
            "Monday": [
                {
                    "hour": 0
                    "listen_count": 26,
                },
                {
                    "hour": 1
                    "listen_count": 30,
                },
                {
                    "hour": 2
                    "listen_count": 4,
                },
                "..."
```

(continues on next page)

(continued from previous page)

```
            ],
            "Tuesday": ["..."],
            "..."
        },
        "stats_range": "all_time",
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
    }
}
```

Note:

- This endpoint is currently in beta

Parameters

- **range** (str) – Optional, time interval for which statistics should be returned, possible values are *ALLOWED_STATISTICS_RANGE*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/year-in-music/**
Get data for year in music stuff

GET /1/stats/user/(*user_name*)**/recordings**
Get top recordings for user `user_name`.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "recordings": [
            {
                "artist_mbids": [],
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "recording_mbid": "0fe11cd3-0be4-467b-84fa-0bd524d45d74",
                "release_mbid": "",
                "release_name": "Delirium (Deluxe)",
                "track_name": "Love Me Like You Do - From \"Fifty Shades of Grey\"
↪"
            },
            {
                "artist_mbids": [],
                "artist_name": "The Fray",
```

(continues on next page)

(continued from previous page)


```
                "listen_count": 23,
                "recording_mbid": "0008ab49-a6ad-40b5-aa90-9d2779265c22",
                "release_mbid": "",
                "release_name": "How to Save a Life",
                "track_name": "How to Save a Life"
            }
        ],
        "count": 2,
        "total_recording_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- We only calculate the top 1000 all_time recordings
- `artist_mbids`, `artist_msid`, `release_name`, `release_mbid`, `release_msid`, `recording_mbid` and `recording_msid` are optional fields and may not be present in all the responses

Parameters

- **`count`** (`int`) – Optional, number of recordings to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **`offset`** (`int`) – Optional, number of recordings to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 recordings will be skipped, defaults to 0
- **`range`** (`str`) – Optional, time interval for which statistics should be returned, possible values are *`ALLOWED_STATISTICS_RANGE`*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`GET /1/stats/user/` (*user_name*) **`/artist-map`**

Get the artist map for user `user_name`. The artist map shows the number of artists the user has listened to from different countries of the world.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "artist_map": [
            {
                "country": "USA",
                "artist_count": 34
            },
            {
                "country": "GBR",
                "artist_count": 69
            },
            {
                "country": "IND",
                "artist_count": 32
            }
        ],
        "stats_range": "all_time"
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
    }
}
```


Note:

- This endpoint is currently in beta
- We cache the results for this query for a week to improve page load times, if you want to request fresh data you can use the `force_recalculate` flag.

Parameters

- **`range`** (`str`) – Optional, time interval for which statistics should be returned, possible values are *`ALLOWED_STATISTICS_RANGE`*, defaults to `all_time`
- **`force_recalculate`** (`bool`) – Optional, recalculate the data instead of returning the cached result.

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`GET /1/stats/user/` (*user_name*) **`/releases`**

Get top releases for user `user_name`.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "releases": [
            {
                "artist_mbids": [],
                "artist_name": "Coldplay",
                "listen_count": 26,
                "release_mbid": "",
                "release_name": "Live in Buenos Aires"
            },
            {
                "artist_mbids": [],
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "release_mbid": "",
                "release_name": "Delirium (Deluxe)"
            },
            {
                "artist_mbids": [],
                "artist_name": "The Fray",
                "listen_count": 25,
                "release_mbid": "",
                "release_name": "How to Save a Life"
            },
        ],
        "count": 3,
        "total_release_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- `artist_mbids`, `artist_msid`, `release_mbid` and `release_msid` are optional fields and may not be present in all the responses

Parameters

- **`count`** (`int`) – Optional, number of releases to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **`offset`** (`int`) – Optional, number of releases to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 releases will be skipped, defaults to 0
- **`range`** (`str`) – Optional, time interval for which statistics should be returned, possible values are *`ALLOWED_STATISTICS_RANGE`*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!

- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/artists**

Get top artists for user `user_name`.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "artists": [
            {
               "artist_mbids": ["93e6118e-7fa8-49f6-9e02-699a1ebce105"],
               "artist_name": "The Local train",
               "listen_count": 385
            },
            {
               "artist_mbids": ["ae9ed5e2-4caf-4b3d-9cb3-2ad626b91714"],
               "artist_name": "Lenka",
               "listen_count": 333
            },
            {
               "artist_mbids": ["cc197bad-dc9c-440d-a5b5-d52ba2e14234"],
               "artist_name": "Coldplay",
               "listen_count": 321
            }
        ],
        "count": 3,
        "total_artist_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- `artist_mbids` and `artist_msid` are optional fields and may not be present in all the responses

Parameters

- **count** (`int`) – Optional, number of artists to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **offset** (`int`) – Optional, number of artists to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 artists will be skipped, defaults to 0

- **range** (str) – Optional, time interval for which statistics should be returned, possible values are *ALLOWED_STATISTICS_RANGE*, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

Status API Endpoints

GET /1/status/get-dump-info

Get information about ListenBrainz data dumps. You need to pass the *id* parameter in a GET request to get data about that particular dump.

Example response:

```
{
    "id": 1,
    "timestamp": "20190625-165900"
}
```

Query Parameters

- **id** – Integer specifying the ID of the dump, if not provided, the endpoint returns information about the latest data dump.

Status Codes

- 200 OK – You have data.
- 400 Bad Request – You did not provide a valid dump ID. See error message for details.
- 404 Not Found – Dump with given ID does not exist.

Response Headers

- Content-Type – *application/json*

User Timeline API Endpoints

These api endpoints allow to create and fetch timeline events for a user.

POST /1/user/(*user_name*)**/timeline-event/create/notification**

Post a message with a link on a user's timeline. Only approved users are allowed to perform this action.

The request should contain the following data:

```
{
    "metadata": {
        "message": "<the message to post, required>",
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Parameters

- **user_name** (str) – The MusicBrainz ID of the user on whose timeline the message is to be posted.

Status Codes

- 200 OK – Successful query, message has been posted!
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 403 Forbidden – Forbidden, you are not an approved user.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

POST /1/user/(*user_name*)**/timeline-event/create/recording**

Make the user recommend a recording to their followers.

The request should post the following data about the recording being recommended:

```
{
    "metadata": {
        "artist_name": "<The name of the artist, required>",
        "track_name": "<The name of the track, required>",
        "recording_msid": "<The MessyBrainz ID of the recording, required>",
        "release_name": "<The name of the release, optional>",
        "recording_mbid": "<The MusicBrainz ID of the recording, optional>"
    }
}
```

Parameters

- **user_name** (str) – The MusicBrainz ID of the user who is recommending the recording.

Status Codes

- 200 OK – Successful query, recording has been recommended!
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 401 Unauthorized – Unauthorized, you do not have permissions to recommend recordings on the behalf of this user
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

POST /1/user/(*user_name*)**/feed/events/delete**

Delete those events from user's feed that belong to them. Supports deletion of recommendation and notification. Along with the authorization token, post the event type and event id. For example:

```
{
    "event_type": "recording_recommendation",
    "id": "<integer id of the event>"
}
```

```
{
    "event_type": "notification",
    "id": "<integer id of the event>"
}
```

Parameters

- **user_name** (str) – The MusicBrainz ID of the user from whose timeline events are being deleted

Status Codes

- 200 OK – Successful deletion
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 401 Unauthorized – Unauthorized
- 404 Not Found – User not found
- 500 Internal Server Error – API Internal Server Error

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/feed/events**

Get feed events for a user's timeline.

Parameters

- **user_name** (str) – The MusicBrainz ID of the user whose timeline is being requested.
- **max_ts** – If you specify a `max_ts` timestamp, events with timestamps less than the value will be returned
- **min_ts** – If you specify a `min_ts` timestamp, events with timestamps greater than the value will be returned
- **count** (int) – Optional, number of events to return. Default: `DEFAULT_ITEMS_PER_GET` . Max: `MAX_ITEMS_PER_GET`

Status Codes

- 200 OK – Successful query, you have feed events!
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 401 Unauthorized – Unauthorized, you do not have permission to view this user's feed.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

Social API Endpoints

These apis allow to interact with social features of ListenBrainz.

GET /1/user/(*user_name*)**/followers**

Fetch the list of followers of the user `user_name`. Returns a JSON with an array of user names like these:

```
{
    "followers": ["rob", "mr_monkey", "..."],
    "user": "shivam-kapila"
}
```

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found

GET /1/user/(*user_name*)**/following**

Fetch the list of users followed by the user `user_name`. Returns a JSON with an array of user names like these:

```
{
    "followers": ["rob", "mr_monkey", "..."],
    "user": "shivam-kapila"
}
```

Status Codes

- 200 OK – Yay, you have data!
- 404 Not Found – User not found

POST /1/user/(*user_name*)**/unfollow**

Unfollow the user `user_name`. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>
- Content-Type – *application/json*

Status Codes

- 200 OK – Successfully unfollowed the user `user_name`.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

POST /1/user/(*user_name*)**/follow**

Follow the user `user_name`. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>
- Content-Type – *application/json*

Status Codes

- 200 OK – Successfully followed the user `user_name`.
- 400 Bad Request –
  - Already following the user `user_name`.
  - Trying to follow yourself.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

Pinned Recording API Endpoints

These API endpoints allow submitting, deleting, and retrieving ListenBrainz pinned recordings for users.

`POST /1/pin/unpin`

Unpins the currently active pinned recording for the user. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – recording unpinned.
- 401 Unauthorized – invalid authorization. See error message for details.
- 404 Not Found – could not find the active recording to unpin for the user. See error message for details.

Response Headers

- Content-Type – *application/json*

`POST /1/pin`

Pin a recording for user. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should contain only one pinned recording item in the payload.

The format of the JSON to be POSTed to this endpoint should look like the following:

```
{
    "recording_msid": "40ef0ae1-5626-43eb-838f-1b34187519bf",
    "recording_mbid": "<this field is optional>",
    "blurb_content": "Wow..",
    "pinned_until": 1824001816
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – feedback accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

POST /1/pin/delete/ (*row_id*)

Deletes the pinned recording with given `row_id` from the server. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>

Parameters

- **row_id** (`int`) – the row_id of the pinned recording that should be deleted.

Status Codes

- 200 OK – recording unpinned.
- 401 Unauthorized – invalid authorization. See error message for details.
- 404 Not Found – the requested row_id for the user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/ (*user_name*) **/pins/following**

Get a list containing the active pinned recordings for all users in a user's `user_name` following list. The returned pinned recordings are sorted in descending order of the time they were pinned. The JSON returned by the API will look like the following:

```
{
    "count": 1,
    "offset": 0,
    "pinned_recordings": [
        {
        "blurb_content": "Spectacular recording!",
        "created": 1624000841,
        "row_id": 1,
        "pinned_until": 1624605641,
        "recording_mbid": null,
        "recording_msid": "40ef0ae1-5626-43eb-838f-1b34187519bf",
        "track_metadata": {
            "artist_name": "Rick Astley",
            "track_name": "Never Gonna Give You Up"
        },
        "user_name": "-- the MusicBrainz ID of the user who pinned this recording␣
↪--"
        },
        "-- more pinned recordings from different users here ---"
    ],
    "user_name": "-- the MusicBrainz ID of the original user --"
}
```

Parameters

- **user_name** (`str`) – the MusicBrainz ID of the user whose followed user's pinned recordings are being requested.

- **count** (int) – Optional, number of pinned recording items to return, Default: DEFAULT_ITEMS_PER_GET Max: MAX_ITEMS_PER_GET
- **offset** (int) – Optional, number of pinned recording items to skip from the beginning, for pagination. Ex. An offset of 5 means the most recent pinned recordings from the first 5 users will be skipped, defaults to 0

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – Invalid query parameters. See error message for details.
- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

GET /1/ (*user_name*) **/pins**

Get a list of all recordings ever pinned by a user with given user_name in descending order of the time they were originally pinned. The JSON returned by the API will look like the following:

```
{
    "count": 10,
    "offset": 0,
    "pinned_recordings": [
        {
            "blurb_content": "Awesome recording!",
            "created": 1623997168,
            "row_id": 10,
            "pinned_until": 1623997485,
            "recording_mbid": null,
            "recording_msid": "fd7d9162-a284-4a10-906c-faae4f1e166b"
            "track_metadata": {
                "artist_name": "Rick Astley",
                "track_name": "Never Gonna Give You Up"
            }
        },
        "-- more pinned recording items here ---"
    ],
    "total_count": 10,
    "user_name": "-- the MusicBrainz ID of the user --"
}
```

Parameters

- **user_name** (str) – the MusicBrainz ID of the user whose pin track history requested.
- **count** (int) – Optional, number of pinned recording items to return, Default: DEFAULT_ITEMS_PER_GET Max: MAX_ITEMS_PER_GET
- **offset** (int) – Optional, number of pinned recording items to skip from the beginning, for pagination. Ex. An offset of 5 means the most recent 5 pinned recordings from the user will be skipped, defaults to 0

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – Invalid query parameters. See error message for details.

- 404 Not Found – The requested user was not found.

Response Headers

- Content-Type – *application/json*

Color API Endpoints

These API endpoints allow fetching releases with recordings and cover art details for a given color.

GET /1/color/(*color*)
Fetch a list of releases that have cover art that has a predominant color that is close to the given color.

```
{
    "payload": {
        "releases" : [
            {
              "artist_name": "Letherette",
              "color": [ 250, 90, 192 ],
              "dist": 109.973,
              "release_mbid": "00a109da-400c-4350-9751-6e6f25e89073",
              "caa_id": 34897349734,
              "release_name": "EP5",
              "recordings": "< array of listen formatted metadata >",
              },
            ". . ."
        ]
    }
}
```

Status Codes

- 200 OK – success

Response Headers

- Content-Type – *application/json*

Rate limiting

The ListenBrainz API is rate limited via the use of rate limiting headers that are sent as part of the HTTP response headers. Each call will include the following headers:

- **X-RateLimit-Limit**: Number of requests allowed in given time window
- **X-RateLimit-Remaining**: Number of requests remaining in current time window
- **X-RateLimit-Reset-In**: Number of seconds when current time window expires (*recommended*: this header is resilient against clients with incorrect clocks)
- **X-RateLimit-Reset**: UNIX epoch number of seconds (without timezone) when current time window expires [#]

Rate limiting is automatic and the client must use these headers to determine the rate to make API calls. If the client exceeds the number of requests allowed, the server will respond with error code `429: Too Many Requests`. Requests that provide the *Authorization* header with a valid user token may receive higher rate limits than those without valid user tokens.

Timestamps

All timestamps used in ListenBrainz are UNIX epoch timestamps in UTC. When submitting timestamps to us, please ensure that you have no timezone adjustments on your timestamps.

Constants

Constants that are relevant to using the API:

`listenbrainz.webserver.views.api_tools.`**`MAX_LISTEN_SIZE = 10240`**
Maximum overall listen size in bytes, to prevent egregious spamming.

`listenbrainz.webserver.views.api_tools.`**`MAX_ITEMS_PER_GET = 100`**
The maximum number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**`DEFAULT_ITEMS_PER_GET = 25`**
The default number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**`MAX_TAGS_PER_LISTEN = 50`**
The maximum number of tags per listen.

`listenbrainz.webserver.views.api_tools.`**`MAX_TAG_SIZE = 64`**
The maximum length of a tag

`listenbrainz.listenstore.`**`LISTEN_MINIMUM_TS = 1033430400`**
The minimum acceptable value for listened_at field

`data.model.common_stat.`**`ALLOWED_STATISTICS_RANGE = ['week', 'month', 'quarter', 'half_yearly`**
list of allowed value for range param accepted by various statistics endpoints

1.2 API Usage Examples

Note: These examples are written in Python version **3.6.3** and use requests version **2.18.4**.

1.2.1 Prerequisites

All the examples assume you have a development version of the ListenBrainz server set up on `localhost`. Remember to set `DEBUG` to `True` in the config. When in production, you can replace `localhost` with `api.listenbrainz.org` to use the real API. In order to use either one, you'll need a token. You can find it under `ROOT/profile/` when signed in, with `ROOT` being either `localhost` for the dev version or `listenbrainz.org` for the real API.

Caution: You should use the token from the API you're using. In production, change the token to one from `listenbrainz.org`.

1.2.2 Examples

Submitting Listens

See *JSON Documentation* for details on the format of the Track dictionaries.

If everything goes well, the json response should be `{"status": "ok"}`, and you should see a recent listen of "Never Gonna Give You Up" when you visit `ROOT/user/{your-user-name}`.

```
from time import time
import requests

ROOT = '127.0.0.1'

def submit_listen(listen_type, payload, token):
    """Submits listens for the track(s) in payload.

    Args:
        listen_type (str): either of 'single', 'import' or 'playing_now'
        payload: A list of Track dictionaries.
        token: the auth token of the user you're submitting listens for

    Returns:
         The json response if there's an OK status.

    Raises:
         An HTTPError if there's a failure.
         A ValueError is the JSON in the response is invalid.
    """

    response = requests.post(
        url="http://{0}/1/submit-listens".format(ROOT),
        json={
            "listen_type": listen_type,
            "payload": payload,
        },
        headers={
            "Authorization": "Token {0}".format(token)
        }
    )

    response.raise_for_status()

    return response.json()


if __name__ == "__main__":
    EXAMPLE_PAYLOAD = [
        {
            # An example track.
            "listened_at": int(time()),
            "track_metadata": {
                "additional_info": {
                    "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
                    "artist_mbids": [
                        "db92a151-1ac2-438b-bc43-b82e149ddd50"
                    ],
                    "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
```

(continues on next page)

(continued from previous page)

```
                    "tags": ["you", "just", "got", "semi", "rick", "rolled"]
                },
                "artist_name": "Rick Astley",
                "track_name": "Never Gonna Give You Up",
                "release_name": "Whenever you need somebody"
            }
        }
    ]

    # Input token from the user and call submit listen
    token = input('Please enter your auth token: ')
    json_response = submit_listen(listen_type='single', payload=EXAMPLE_PAYLOAD,␣
↪token=token)

    print("Response was: {0}".format(json_response))
    print("Check your listens - there should be a Never Gonna Give You Up track,␣
↪played recently.")
```

Getting Listen History

See *JSON Documentation* for details on the format of the Track dictionaries.

If there's nothing in the listen history of your user, you can run `submit_listens` before this.

If there is some listen history, you should see a list of tracks like this:

```
import requests

ROOT = '127.0.0.1'
# The following token must be valid, but it doesn't have to be the token of the user␣
↪you're
# trying to get the listen history of.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}


def get_listens(username, min_ts=None, max_ts=None, count=None):
    """Gets the listen history of a given user.

    Args:
        username: User to get listen history of.
        min_ts: History before this timestamp will not be returned.
                DO NOT USE WITH max_ts.
        max_ts: History after this timestamp will not be returned.
                DO NOT USE WITH min_ts.
        count: How many listens to return. If not specified,
               uses a default from the server.

    Returns:
        A list of listen info dictionaries if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
```

(continues on next page)

(continued from previous page)

```
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/user/{1}/listens".format(ROOT, username),
        params={
            "min_ts": min_ts,
            "max_ts": max_ts,
            "count": count,
        },
        # Note that an authorization header isn't compulsary for requests to get␣
↪listens
        # BUT requests with authorization headers are given relaxed rate limits by␣
↪ListenBrainz
        headers=AUTH_HEADER,
    )

    response.raise_for_status()

    return response.json()['payload']['listens']

if __name__ == "__main__":
    username = input('Please input the MusicBrainz ID of the user: ')
    listens = get_listens(username)

    for track in listens:
        print("Track: {0}, listened at {1}".format(track["track_metadata"]["track_name
↪"],
                                                   track["listened_at"]))
```

```
Track: Never Gonna Give You Up, listened at 1512040365
Track: Never Gonna Give You Up, listened at 1511977429
Track: Never Gonna Give You Up, listened at 1511968583
Track: Never Gonna Give You Up, listened at 1443521965
Track: Never Gonna Give You Up, listened at 42042042
```

Latest Import

Set and get the timestamp of the latest import into ListenBrainz.

Setting

```
from time import time
import requests

ROOT = '127.0.0.1'

def set_latest_import(timestamp, token, service="lastfm"):
    """Sets the time of the latest import.

    Args:
        timestamp: Unix epoch to set latest import to.
        token: the auth token of the user you're setting latest_import of
        service: service to set latest import time of.
```

(continues on next page)

(continued from previous page)

```
    Returns:
        The JSON response if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON response is invalid.
    """
    response = requests.post(
        url="http://{0}/1/latest-import".format(ROOT),
        json={
            "ts": timestamp,
            "service": service
        },
        headers={
            "Authorization": "Token {0}".format(token),
        }
    )

    response.raise_for_status()

    return response.json()

if __name__ == "__main__":
    ts = int(time())
    token = input('Please enter your auth token: ')
    json_response = set_latest_import(ts, token)

    print("Response was: {0}".format(json_response))
    print("Set latest import time to {0}.".format(ts))
```

Getting

If your user has never imported before and the latest import has never been set by a script, then the server will return `0` by default. Run `set_latest_import` before this if you don't want to actually import any data.

```
import requests

ROOT = '127.0.0.1'
# The token can be any valid token.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}

def get_latest_import(username, service="lastfm"):
    """Gets the latest import timestamp of a given user.

    Args:
        username: User to get latest import time of.
        service: service to get latest import time of.

    Returns:
        A Unix timestamp if there's an OK status.
```

(continues on next page)

(continued from previous page)

```
    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/latest-import".format(ROOT),
        params={
            "user_name": username,
            "service": service
        },
        headers=AUTH_HEADER,
    )

    response.raise_for_status()
    return response.json()["latest_import"]


if __name__ == "__main__":
    username = input('Please input the MusicBrainz ID of the user: ')
    timestamp = get_latest_import(username)

    print("User {0} last imported on {1}".format(username, timestamp))
```

You should see output like this:

```
User naiveaiguy last imported on 30 11 2017 at 12:23
```

1.3 JSON Documentation

Note: Do not submit copyrighted information in these fields!

1.3.1 Submission JSON

To submit a listen via our API (see: *ListenBrainz API*), `POST` a JSON document to the `submit-listens` endpoint. Submit one of three types JSON documents:

- `single`: Submit single listen
  - Indicates user just finished listening to track
  - `payload` should contain information about *exactly one* track
- `playing_now`: Submit `playing_now` notification
  - Indicates that user just began listening to track
  - `payload` should contain information about *exactly one* track
  - Submitting `playing_now` documents is optional
  - Timestamp must be omitted from a `playing_now` submission.
- `import`: Submit previously saved listens

- `payload` should contain information about *at least one* track
- submitting multiple listens in one go is allowed, but the complete JSON document may not exceed `MAX_LISTEN_SIZE` bytes in size

The `listen_type` element defines different types of submissions. The element is placed at the top-most level of the JSON document. The only other required element is the `payload` element. This provides an array of listens – the payload may be one or more listens (as designated by `listen_type`):

```
{
  "listen_type": "single",
  "payload": [
      "--- listen data here ---"
  ]
}
```

A sample listen payload may look like:

```
{
  "listened_at": 1443521965,
  "track_metadata": {
    "additional_info": {
      "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
      "artist_mbids": [
        "db92a151-1ac2-438b-bc43-b82e149ddd50"
      ],
      "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
      "tags": [ "you", "just", "got", "rick rolled!"]
    },
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
    "release_name": "Whenever you need somebody"
  }
}
```

A complete submit listen JSON document may look like:

```
{
  "listen_type": "single",
  "payload": [
    {
      "listened_at": 1443521965,
      "track_metadata": {
        "additional_info": {
          "media_player": "Rhythmbox",
          "submission_client": "Rhythmbox ListenBrainz Plugin",
          "submission_client_version": "1.0",
          "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
          "artist_mbids": [
            "db92a151-1ac2-438b-bc43-b82e149ddd50"
          ],
          "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
          "tags": [ "you", "just", "got", "rick rolled!"]
        },
        "artist_name": "Rick Astley",
        "track_name": "Never Gonna Give You Up",
        "release_name": "Whenever you need somebody"
      }
```

(continues on next page)

(continued from previous page)

```
    }
  ]
}
```

1.3.2 Fetching listen JSON

The JSON documents returned from our API look like the following:

```
{
  "payload": {
    "count": 25,
    "user_id": "-- the MusicBrainz ID of the user --",
    "listens": [
      "-- listen data here ---"
    ]
  }
}
```

The number of listens in the document are returned by the top-level `count` element. The `user_id` element contains the MusicBrainz ID of the user whose listens are being returned. The other element is the `listens` element. This is a list which contains the listen JSON elements (described above).

The JSON document returned by the API endpoint for getting tracks being played right now is the same as above, except that it also contains the `payload/playing_now` element as a boolean set to True.

1.3.3 Payload JSON details

A minimal payload must include `track_metadata/artist_name` and `track_metadata/track_name` elements:

```
{
  "track_metadata": {
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
  }
}
```

`artist_name` and `track_name` elements must be simple strings.

The payload should also include the `listened_at` element, which must be an integer representing the Unix time when the track was listened to. The minimum accepted value for this field is `LISTEN_MINIMUM_TS`. playing_now requests should not have a `listened_at` field

Add additional metadata you may have for a track to the `additional_info` element. Any additional information allows us to better correlate your listen data to existing MusicBrainz-based data. If you have MusicBrainz IDs available, submit them!

The following optional elements may also be included in the `track_metadata` element:

element	description
`release_name`	the name of the release this recording was played from.

The following optional elements may also be included in the `additional_info` element. If you do not have the data for any of the following fields, omit the key entirely:

Table 1: Title

element	description
`artist_mbids`	A list of MusicBrainz Artist IDs, one or more Artist IDs may be included here. If you have a complete MusicBrainz artist credit that contains multiple Artist IDs, include them all in this list.
`release_group_mbid`	A MusicBrainz Release Group ID of the release group this recording was played from.
`release_mbid`	A MusicBrainz Release ID of the release this recording was played from.
`recording_mbid`	A MusicBrainz Recording ID of the recording that was played.
`track_mbid`	A MusicBrainz Track ID associated with the recording that was played.
`work_mbids`	A list of MusicBrainz Work IDs that may be associated with this recording.
`tracknumber`	The tracknumber of the recording. This first recording on a release is tracknumber 1.
`isrc`	The ISRC code associated with the recording.
`spotify_id`	The Spotify track URL associated with this recording. e.g.: http://open.spotify.com/track/1rrgWMXGCGHru5bIRxGFV0
`tags`	A list of user-defined folksonomy tags to be associated with this recording. For example, you have apply tags such as `punk`, `see-live`, `smelly`. You may submit up to `MAX_TAGS_PER_LISTEN` tags and each tag may be up to `MAX_TAG_SIZE` characters large.
`media_player`	The name of the program being used to listen to music. Don't include a version number here.
`media_player_version`	The version of the program being used to listen to music.
`submission_client`	The name of the client that is being used to submit listens to ListenBrainz. If the media player has the ability to submit listens built-in then this value may be the same as `media_player`. Don't include a version number here.
`submission_client_version`	The version of the submission client.
`music_service`	If the song being listened to comes from an online service, the canonical domain of this service (see below for more details).
`music_service_name`	If the song being listened to comes from an online service and you don't know the canonical domain, a name that represents the service.
`origin_url`	If the song of this listen comes from an online source, the URL to the place where it is available. This could be a spotify url (see `spotify_id`), a YouTube video URL, a Soundcloud recording page URL, or the full URL to a public MP3 file. If there is a webpage for this song (e.g. Youtube page, Soundcloud page) **do not** try and resolve the URL to an actual audio resource.

Note: Music service names

The `music_service` field should be a domain name rather than a textual description or URL. This allows us to refer unambiguously to a service without worrying about capitalization or full/short names (such as the difference between "Internet Archive", "The Internet Archive" or "Archive"). If we use this data on ListenBrainz, we will perform a mapping from the domain name to a canonical name. Below is an example of mappings that we currently support. If you are submitting from a service which doesn't appear in this list, you should determine a canonical domain from the domain of the service. Only if you cannot determine a domain for the service should you use the text-only `music_service_name` field.

Table 2: Music services domain/name mapping

domain	name
spotify.com	Spotify
bandcamp.com	Bandcamp
youtube.com	YouTube
music.youtube.com	YouTube Music
deezer.com	Deezer
tidal.com	TIDAL
music.apple.com	Apple Music
archive.org	Internet Archive
soundcloud.com	Soudcloud
jamendo.com	Jamendo Music
play.google.com	Google Play Music

1.3.4 Client Metadata examples

Here are a few examples of how to fill in the `media_player`, `submission_client` and `music_service` fields based on our current recommendations.

BrainzPlayer on the ListenBrainz website playing a video from YouTube

```
{
  "track_metadata": {
      "additional_info": {
          "media_player": "BrainzPlayer",
          "music_service": "youtube.com",
          "origin_url": "https://www.youtube.com/watch?v=JKFBiaoFHoY",
          "submission_client": "BrainzPlayer"
      },
      "artist_name": "Mdou Moctar",
      "release_name": "Ilana (The Creator)",
      "track_name": "Inizgam"
  }
}
```

BrainzPlayer on the ListenBrainz website playing a video from Spotify

Note that even though the `origin_url` is `https://open.spotify.com`, we set `music_service` to spotify.com (see above note).

```
{
  "track_metadata": {
      "additional_info": {
          "media_player": "BrainzPlayer",
          "music_service": "spotify.com",
          "origin_url": "https://open.spotify.com/track/5fEjp2F0Sqr9fMuLSaDqz0",
          "submission_client": "BrainzPlayer"
      },
      "artist_name": "Les Filles de Illighadad",
```

(continues on next page)

(continued from previous page)

```
        "release_name": "Eghass Malan",
        "track_name": "Inssegh Inssegh"
    }
}
```

Using Otter for Funkwhale on android, and submitting with Simple Scrobbler

In this case, the media player and submission client are completely separate programs. Because music is being played from a user's private collection and not a streaming service, don't include music_service or origin_url.

```
{
  "track_metadata": {
        "additional_info": {
            "media_player": "Otter",
            "media_player_version": "1.0.21",
            "submission_client": "Simple Scrobbler"
            "submission_client_version": "1.7.0"
        },
        "artist_name": "Les Filles de Illighadad",
        "release_name": "Eghass Malan",
        "track_name": "Inssegh Inssegh"
    }
}
```

Rhythmbox player listening to Jamendo

```
{
  "track_metadata": {
        "additional_info": {
            "media_player": "Rhythmbox",
            "music_service": "jamendo.com",
            "music_service_name": "Jamendo Music"
            "origin_url": "https://www.jamendo.com/track/1466090/universal-funk",
            "submission_client": "Rhythmbox ListenBrainz Plugin"
        },
        "artist_name": "Duo Teslar",
        "track_name": "Universal Funk"
    }
}
```

Listening to a recording from Bandcamp and submitting with the browser extension WebScrobbler

Because playback happens in the browser, there is no specific `media_player`.

```
{
        "track_metadata": {
                "additional_info": {
                        "music_service": "bandcamp.com",
                        "music_service_name": "Bandcamp",
                        "submission_client": "WebScrobbler",
                        "submission_client_version": "v2.48.0"
                        "origin_url": "https://greencookierecords.bandcamp.com/track/
↪shake",
```

(continues on next page)

(continued from previous page)

```
                },
                "artist_name": "I Mitomani Beat",
                "release_name": "Fuori Dal Tempo",
                "track_name": "Shake",
        }
}
```

At this point, we are not removing any other elements that may be submitted via the `additional_info` element. We're open to see how people will make use of these unspecified fields and may decide to formally specify or scrub elements in the future.

1.4 Client Libraries

Client Libraries have already been written by the community for some languages.

1.4.1 Haskell

- listenbrainz-client

1.4.2 Go

- go-listenbrainz

1.4.3 Rust

- listenbrainz-rust

1.4.4 .NET

- MetaBrainz.ListenBrainz

1.4.5 Python

- pylistenbrainz

1.5 Last.FM Compatible API for ListenBrainz

There are two versions of the Last.FM API used by clients to submit data to Last.FM.

1. The latest Last.FM API
2. The AudioScrobbler API v1.2

ListenBrainz can understand requests sent to both these APIs and use their data to import listens submitted by clients like VLC and Spotify. Existing Last.FM clients can be pointed to the ListenBrainz proxy URL and they should submit listens to ListenBrainz instead of Last.FM.

Note: This information is also present on the ListenBrainz website.

1.5.1 AudioScrobbler API v1.2

Clients supporting the old version of the AudioScrobbler API (such as VLC and Spotify) can be configured to work with ListenBrainz by making the client point to `http://proxy.listenbrainz.org` and using your MusicBrainz ID as username and the LB Authorization Token as password.

If the software you are using doesn't support changing where the client submits info (like Spotify), you can edit your `/etc/hosts` file as follows:

```
138.201.169.196    post.audioscrobbler.com
138.201.169.196    post2.audioscrobbler.com
```

1.5.2 Last.FM API

These instructions are for setting up usage of the Last.FM API for Audacious client on Ubuntu. These steps can be modified for other clients as well.

For development

1. Install dependencies from here, then clone the repo and install audacious.
2. Before installing audacious-plugins, edit the file *audacious-plugins/src/scrobbler2/scrobbler.h* to update the following setting on line L28. This is required only because the local server does not have https support.:

   ```
   `SCROBBLER_URL` to "http://ws.audioscrobbler.com/2.0/".
   ```

3. Compile and install the plugins from the instructions given here.
4. Edit the `/etc/hosts` file and add the following entry:

   ```
   127.0.0.1 ws.audioscrobbler.com
   ```

5. Flush dns and restart network manager using:

   ```
   $ sudo /etc/init.d/dns-clean start
   $ sudo /etc/init.d/networking restart
   ```

6. Register an application on MusicBrainz with the following Callback URL `http://<HOSTURL>/login/musicbrainz/post` and update the received MusicBrainz Client ID and Client Secret in config.py of ListenBrainz. `HOSTURL` should be as per the settings of the server. Example: `localhost`
7. In Audacious, go to File > Settings > Plugins > Scrobbler2.0 and enable it. Now open its settings and then authenticate.
8. **When you get a URL from your application which look like this `http://last.fm/api/auth/?api_key=as3..234`**

   - If you are running a local server, then `HOSTURL` should be similar to "localhost:7080".
   - If you are not running the server, then `HOSTURL` should be "api.listenbrainz.org".

For users

1. Repeat all the above steps, except for steps 2 and 6.
2. For Step 8, choose the 2nd option for `HOSTURL`.

1.6 ListenBrainz Data Dumps

ListenBrainz provides data dumps that you can import into your own server or use for other purposes. The full data dumps are created twice a month and the incremental data dumps twice a week. Each dump contains a number of different files. Depending on your use cases, you may or may not require all of them.

We have a bunch of *commands* which may be useful in interacting with dumps during local development as well.

1.6.1 Dump mirrors

See the ListenBrainz data page for information about where to download the data dumps from.

1.6.2 File Descriptions

A ListenBrainz data dump consists of three archives:

1. `listenbrainz-public-dump.tar.xz`
2. `listenbrainz-listens-dump.tar.xz`
3. `listenbrainz-listens-dump-spark.tar.xz`

listenbrainz-public-dump.tar.xz

This file contains information about ListenBrainz users and statistics derived from listens submitted to ListenBrainz calculated from users, artists, recordings etc.

listenbrainz-listens-dump.tar.xz

This is the core ListenBrainz data dump. This file contains all the listens submitted to ListenBrainz by its users.

listenbrainz-listens-dump-spark.tar.xz

This is also a dump of the core ListenBrainz listen data. These dumps are made for consumption by the ListenBrainz Apache Spark cluster, formatting all listens into monthly JSON files that can easily be loaded into dataframes.

1.6.3 Structure of the listens dump

The ListenBrainz listen dump consists of listens broken down by year and month. At the top level there are directories for each of the year for which we have data. Inside each year there are listens files with month number as its name:

1. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/1.listens`
2. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/2.listens`
3. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/3.listens`
4. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/4.listens`
5. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/5.listens`

Each of the .listens files contains one JSON document per line – each of the JSON documents is one listen, formatted in the standard listens format.

1.6.4 Incremental dumps (BETA)

> **Warning:** The incremental dumps are in beta. We know of some data consistency issues where incremental dumps have fewer listens than they should. Make sure you use the full dumps if data accuracy is important.

ListenBrainz provides incremental data dumps that you can use to keep up to date with the ListenBrainz dataset without needing to download the full dumps everytime. These dumps have the same structure as the corresponding full dumps, but only contain data that has been submitted since the creation of the previous dump. We create incremental data dumps twice a week.

The basic idea here is that dumps create a linear timeline of the dataset based on the time of submission of data. In order to use the incremental dumps, you must start with the latest full dump and then, applying all incremental dumps since will give you the latest data. The series is consistent, if you take a full dump and apply all incremental dumps since that full dump until the next full dump, you will have the same data as the next full dump.

1.7 Set up ListenBrainz Server development environment

To contribute to the ListenBrainz project, you need a development environment. With your development environment, you can test your changes before submitting a patch to the project. This guide helps you set up a development environment and run ListenBrainz locally on your workstation. By the end of this guide, you will have...

- Installed system dependencies
- Registered a MusicBrainz application
- Initialized development databases
- Running ListenBrainz Server

1.7.1 Clone listenbrainz-server

ListenBrainz is hosted on GitHub at https://github.com/metabrainz/listenbrainz-server/. You can use `git` to clone it to your computer

```
git clone https://github.com/metabrainz/listenbrainz-server.git
```

1.7.2 Install docker

ListenBrainz uses Docker for development. This helps you to easily create your development environment. Therefore, to work on the project, you first need to install Docker. If you haven't already, follow the docker installation instructions for your platform.

1.7.3 Register a MusicBrainz application

Next, you need to register your application and get an OAuth token from MusicBrainz. This allows you to sign into your development environment with your MusicBrainz account.

To register, visit the MusicBrainz applications page. There, look for the option to register your application. Fill out the form with the following data:

- **Name**: (any name that you want and will recognize, e.g. `listenbrainz-server-devel`)
- **Type**: `Web Application`
- **Callback URL**: `http://localhost/login/musicbrainz/post/`

After entering this information, you'll have an OAuth client ID and OAuth client secret. You'll use these for configuring ListenBrainz.

Update config.py

With your new client ID and secret, update the ListenBrainz configuration file. If this is your first time configuring ListenBrainz, copy the sample to a live configuration.

```
cp listenbrainz/config.py.sample listenbrainz/config.py
```

Next, open the file with your favorite text editor and look for this section.

```
# MusicBrainz OAuth
MUSICBRAINZ_CLIENT_ID = "CLIENT_ID"
MUSICBRAINZ_CLIENT_SECRET = "CLIENT_SECRET"
```

Update the strings with your client ID and secret. After doing this, your ListenBrainz development environment is able to authenticate and log in from your MusicBrainz login.

To use the Last.fm importer you need an API account at Last.fm. You can register for one at the Last.fm API page. Look for the following section in `config.py`.

```
# Lastfm API
LASTFM_API_URL = "https://ws.audioscrobbler.com/2.0/"
LASTFM_API_KEY = "USE_LASTFM_API_KEY"
```

Update the LASTFM_API_KEY field with your Last.fm API key.

You also need to update the API_URL field value to http://localhost.

To use the Spotify importer you need to register an application on the Spotify Developer Dashboard. Use http://localhost/profile/music-services/spotify/callback/ as the callback URL.

After that, fill out the Spotify client ID and client secret in the following section of the file.

```
# SPOTIFY
SPOTIFY_CLIENT_ID = ''
SPOTIFY_CLIENT_SECRET = ''
```

Note: The hostname on the callback URL must be the same as the host you use to access your development server. If you use something other than localhost, you should update the SPOTIFY_CALLBACK_URL field accordingly.

To use the CritiqueBrainz reviewer, you'll need to visit the CritiqueBrainz applications page and create/register an application. Use http://localhost/ as the homepage URL and http://localhost/profile/music-services/critiquebrainz/callback/ as the callback URL.

After registering, update the CritiqueBrainz section of the file with the client ID and client secret you obtained.

```
# CRITIQUEBRAINZ
CRITIQUEBRAINZ_CLIENT_ID = ''
CRITIQUEBRAINZ_CLIENT_SECRET = ''
CRITIQUEBRAINZ_REDIRECT_URI = 'http://localhost/profile/music-services/critiquebrainz/
↪callback/'
```

Note: Again, if you use something other than localhost as the host you use to access your development server, you should update the homepage and Authorization callback URL fields accordingly when registering on CritiqueBrainz.

1.7.4 Initialize ListenBrainz containers

Next, run

```
./develop.sh build
```

in the root of the repository. Using docker-compose, this will build multiple Docker images for the different services that make up the ListenBrainz server.

The first time you run this script it might take some time while it downloads all of the required dependencies and builds the services.

1.7.5 Initialize ListenBrainz databases

Your development environment needs some specific databases to work. Before proceeding, run these commands to initialize the databases.

```
./develop.sh manage init_db --create-db
./develop.sh manage init_msb_db --create-db
./develop.sh manage init_ts_db --create-db
```

Your development environment is now ready. Now, let's actually see ListenBrainz load locally!

1.7.6 Run the magic script

Now that the databases are initialized, you can start your development environment by running `develop.sh up`.

```
./develop.sh up
```

Note: By default, the web service listens on port 7000. If you already have a service listening on this port, then you can change it by updating the ports section of `docker/docker-compose.yml`.

```
ports:
- "7000:80"
```

To change the listening port, change only the value before the ":" to the port of your choice and point your browser to `http://localhost:<Port>`

You will see the output of `docker-compose`. You can shut down listenbrainz by pressing CTRL^C. Once everything is running, visit your new site in a browser!

```
http://localhost
```

Now, you are all set to begin making changes and seeing them in real-time inside of your development environment. If you make changes to python code, the server will be automatically restarted. If you make changes to javascript code it will be automatically compiled.

Look at the develop.sh documentation for more details.

1.7.7 Listenbrainz containers

A listenbrainz development environment contains a number of different containers running different services. We provide a small description of each container here:

- `db`: A PostgreSQL server that contains data about users
- `redis`: A redis server to store temporary server data
- `timescale`: A PostgreSQL server with the TimescaleDB extension that stores users listens
- `rabbitmq`: Used for passing listens between different services
- `web`: This is the main ListenBrainz server
- `api_compat`: A Last.fm-compatible API server
- `websockets`: A websocket server used for the user-following and playlist updates on the front-end
- `static_builder`: A helper service to build Javascript/Typescript and CSS assets if they are changed

Note: If you add new python dependencies to ListenBrainz by adding them to `requirements.txt` you will have rebuild the web server. Use

```
./develop.sh build web
```

to do this.

If you add new Javascript dependencies you will have to rebuild the `static_builder`:

```
./develop.sh build static_builder
```

1.7.8 Test your changes with unit tests

Unit tests are an important part of ListenBrainz. It helps make it easier for developers to test changes and help prevent easily avoidable mistakes later on. Before committing new code or making a pull request, run the unit tests on your code.

```
./test.sh
```

This builds and runs the containers needed for the tests. This script configures test-specific data volumes so that test data is isolated from your development data.

To run tests faster, you can use some options to start up the test infrastructure once so that subsequent running of the tests is faster:

```
./test.sh -u # build unit test containers, start up and initialise the database
./test.sh    # run tests, do this as often as you need to
./test.sh -s # stop test containers, but don't remove them
./test.sh -d # stop and remove all test containers
```

If you made any changes to the frontend, you can run the tests for frontend using

```
./test.sh fe
```

You can also make use of the following frontend testing options for efficient testing.

```
./test.sh fe             run frontend tests
./test.sh fe -u          run frontend tests, update snapshots
./test.sh fe -b          build frontend test containers
./test.sh fe -t          run type-checker
```

Also, run the **integration tests** for ListenBrainz.

```
./test.sh int
```

When the tests complete, you will see if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

1.7.9 Lint your code

ListenBrainz uses ESLint to lint the frontend codebase as part of the development process, in Webpack.

ESLint will automatically fix trivial issues and list all other issues in your terminal. Make sure to fix any error with the code you've modified.

There can be quite a lot of logs in the terminal, so if you want to look only at front-end build output, you can use this command to inspect only the static_builder logs:

```
./develop.sh logs -f static_builder
```

1.7.10 Using develop.sh

We provide a utility to wrap docker-compose and some common development processes.

To open a psql session to the listenbrainz database, run:

```
./develop.sh psql
```

To open a psql session to the timescale database containing user listens, run:

```
./develop.sh timescale
```

To open a bash shell in the webserver container, run:

```
./develop.sh bash
```

To open flask shell in the webserver container using ipython with the listenbrainz app loaded, run:

```
./develop.sh shell
```

To open a redis shell:

```
./develop.sh redis
```

`develop.sh` provides a direct interface to invoke manage.py inside a docker container. manage.py is a click script containing a number of listenbrainz management commands. To invoke manage.py, run:

```
./develop.sh manage <command>
```

To get a list of manage.py commands, run:

```
./develop.sh manage --help
```

To pass any other command to docker-compose, run:

```
./develop.sh <command>
```

To get a list of valid docker-compose commands, see the output of `docker-compose help`:

```
./develop.sh help
```

1.8 Set up ListenBrainz Spark development environment

There are two distinct part of the ListenBrainz development environment:

1. listenbrainz – the actual webserver components of ListenBrainz
2. listenbrainz_spark – the spark environment used for features that involve data processing (stats, recommendations etc.)

If you're just working on adding a feature to the ListenBrainz webserver, you **do not** need to set up the Spark development environment. However, if you're looking to add a new stat or improve our fledgling recommender system, you'll need both the webserver and the spark development environment.

This guide should explain how to develop and test new features for ListenBrainz that use Spark.

1.8.1 Set up the webserver

The spark environment is dependent on the webserver. Follow the steps in the *guide to set up the webserver environment*.

Create listenbrainz_spark/config.py

The spark environment needs a config.py in the listenbrainz_spark/ dir. Create it by copying from the sample config file.

```
cp listenbrainz_spark/config.py.sample listenbrainz_spark/config.py
```

1.8.2 Initialize ListenBrainz Spark containers

Run the following command to build the spark containers.

```
./develop.sh spark build
```

The first time you build the containers, you also need to format the `namenode` container.

```
./develop.sh spark format
```

Note: You can run `./develop.sh spark format` any time that you want to delete all of the data that is loaded in spark. This will shut down the spark docker cluster, remove the docker volumes used to store the data, and recreate the HDFS filesystem.

Your development environment is now ready. Now, let's actually see ListenBrainz Spark in action!

1.8.3 Bring containers up

First, ensure that you are running the main ListenBrainz development environment:

```
./develop.sh up
```

Start the ListenBrainz Spark environment:

```
./develop.sh spark up
```

This will also bring up the spark reader container which is described in detail *here*.

1.8.4 Import data into the spark environment

We provide small data dumps that are helpful for working with real ListenBrainz data. Download and import a data dump into your spark environment using the following commands in a separate terminal.

```
./develop.sh spark run spark_reader python manage.py spark request_import_incremental
```

Now, you are all set to begin making changes and seeing them in real-time inside of your development environment!

Once you are done with your work, shut down the containers using the following command.

```
./develop.sh spark down
```

Note: You'll need to run `./develop.sh spark down` every time you restart your environment, otherwise hadoop errors out.

1.8.5 Working with request_consumer

The ListenBrainz webserver and spark cluster interact with each other via the request consumer. For a more detailed guide on working with the request consumer, read this *document*.

1.8.6 Test your changes with unit tests

Unit tests are an important part of ListenBrainz Spark. It helps make it easier for developers to test changes and help prevent easily avoidable mistakes later on. Before committing new code or making a pull request, run the unit tests on your code.

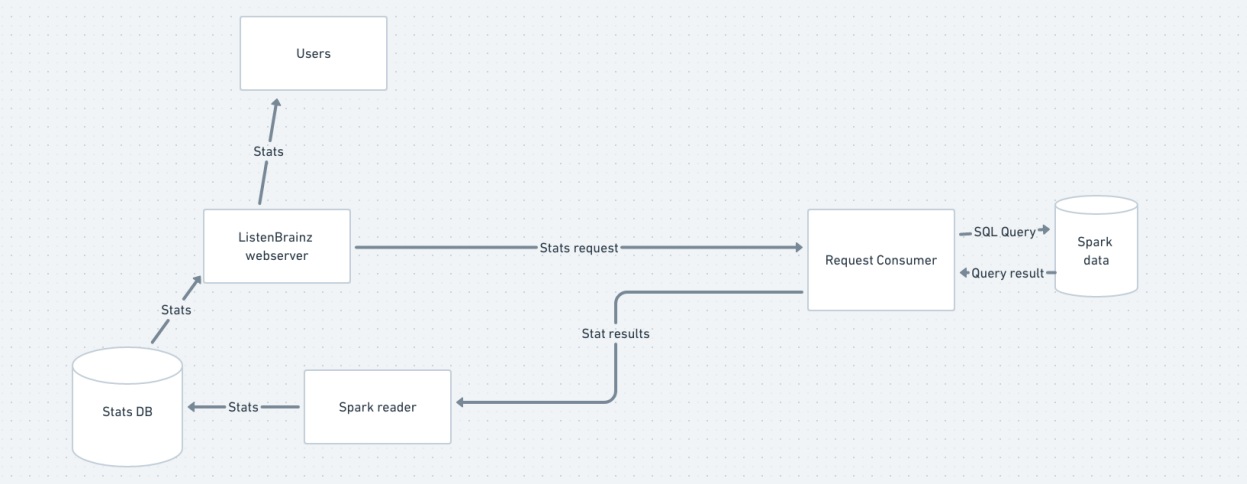
```
./test.sh spark
```

This builds and runs the containers needed for the tests. This script configures test-specific data volumes so that test data is isolated from your development data.

When the tests complete, you will see if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

1.9 ListenBrainz Spark Architecture

In order to actually build features that use Spark, it is important to understand how the ListenBrainz webserver and the Spark environment interact.



The ListenBrainz webserver and Spark cluster are completely seperate entities, only connected by RabbitMQ. This document explains how they interact with each other, taking the example of a stat.

The ListenBrainz environment sends a request to the *request_consumer* script via RabbitMQ. The request consumer, which is connected to Spark, takes the request and uses Spark to compute an appropriate response (or many responses). The request consumer then sends these responses via RabbitMQ to the *spark_reader* script, which runs alongside the webserver. The spark reader then takes the responses, and in the case of a stat, writes them to the ListenBrainz PostgreSQL database. Now that the stat has been updated in the database, users can view them on listenbrainz.org or via the API.

1.9.1 Developing request_consumer

Start the webserver

```
./develop.sh up
```

Start the spark containers

Follow the *instructions* to set up a Spark environment and import a small incremental dump so that you have some data.

Start the spark reader

The spark reader is brought up when you run `./develop.sh spark up`. Now, you have everything needed to work with Spark. You can trigger a request like this

```
./develop.sh manage spark request_user_stats --type=entity --range=week --
↪entity=artists
```

1.10 ListenBrainz Scripts

We have a bunch of python scripts to execute common tasks.

Note: During development, you can use `./develop.sh manage ...` to execute the commands. In production, the command should be run inside the appropriate container using `python manage.py ...`.

1.10.1 ListenBrainz

These commands are helpful in running a ListenBrainz development instance and some other miscellaneous tasks.

./develop.sh manage

```
./develop.sh manage [OPTIONS] COMMAND [ARGS]...
```

init_db

Initializes database.

This process involves several steps:

1. Table structure is created.
2. Primary keys and foreign keys are created.
3. Indexes are created.

```
./develop.sh manage init_db [OPTIONS]
```

Options

-f, --force
Drop existing database and user.

--create-db
Create the database and user.

init_msb_db

Initializes database.

This process involves several steps:

1. Table structure is created.
2. Primary keys and foreign keys are created.
3. Indexes are created.

```
./develop.sh manage init_msb_db [OPTIONS]
```

Options

-f, --force
Drop existing database and user.

--create-db
Skip creating database and user. Tables/indexes only.

init_ts_db

Initializes database.

This process involves several steps:

1. Table structure is created.
2. Indexes are created.
3. Views are created

```
./develop.sh manage init_ts_db [OPTIONS]
```

Options

-f, --force
Drop existing database and user.

--create-db
Create the database and user.

listen-add-userid

Fill in the listen.user_id field based on user_name.

```
./develop.sh manage listen-add-userid [OPTIONS]
```

notify_yim_users

```
./develop.sh manage notify_yim_users [OPTIONS]
```

recalculate_all_user_data

Recalculate all user timestamps and listen counts.

Note: ONLY USE THIS WHEN YOU KNOW WHAT YOU ARE DOING!

```
./develop.sh manage recalculate_all_user_data [OPTIONS]
```

refresh_continuous_aggregates

Update the continuous aggregates in timescale.

```
./develop.sh manage refresh_continuous_aggregates [OPTIONS]
```

run_api_compat_server

```
./develop.sh manage run_api_compat_server [OPTIONS]
```

Options

-h, --host <host>

Default 0.0.0.0

-p, --port <port>

Default 7080

-d, --debug

Turns debugging mode on or off. If specified, overrides 'DEBUG' value in the config file.

run_websockets

```
./develop.sh manage run_websockets [OPTIONS]
```

Options

-h, --host <host>

Default 0.0.0.0

-p, --port <port>

Default 7082

-d, --debug
Turns debugging mode on or off. If specified, overrides 'DEBUG' value in the config file.

set_rate_limits

```
./develop.sh manage set_rate_limits [OPTIONS] PER_TOKEN_LIMIT PER_IP_LIMIT
                                    WINDOW_SIZE
```

Arguments

PER_TOKEN_LIMIT
Required argument

PER_IP_LIMIT
Required argument

WINDOW_SIZE
Required argument

submit-release

Submit a release from MusicBrainz to the local ListenBrainz instance

Specify -u to use the token of this user when submitting, or -t to specify a specific token.

```
./develop.sh manage submit-release [OPTIONS] RELEASEMBID
```

Options

-u, --user <user>

-t, --token <token>

Arguments

RELEASEMBID
Required argument

update_user_emails

```
./develop.sh manage update_user_emails [OPTIONS]
```

1.10.2 Dump Manager

These commands are used to export and import dumps.

./develop.sh manage dump

```
./develop.sh manage dump [OPTIONS] COMMAND [ARGS]...
```

check_dump_ages

Check to make sure that data dumps are sufficiently fresh. Send mail if they are not.

```
./develop.sh manage dump check_dump_ages [OPTIONS]
```

create_feedback

Create a spark formatted dump of user/recommendation feedback data.

```
./develop.sh manage dump create_feedback [OPTIONS]
```

Options

-l, --location <location>
path to the directory where the dump should be made

-t, --threads <threads>
the number of threads to be used while compression

create_full

Create a ListenBrainz data dump which includes a private dump, a statistics dump and a dump of the actual listens from the listenstore.

Args: location (str): path to the directory where the dump should be made threads (int): the number of threads to be used while compression dump_id (int): the ID of the ListenBrainz data dump do_listen_dump: If True, make a listens dump do_spark_dump: If True, make a spark listens dump do_db_dump: If True, make a public/private postgres/timescale dump

```
./develop.sh manage dump create_full [OPTIONS]
```

Options

-l, --location <location>
path to the directory where the dump should be made

-t, --threads <threads>
the number of threads to be used while compression

--dump-id <dump_id>
the ID of the ListenBrainz data dump

--listen, --no-listen

--spark, --no-spark

--db, --no-db
flag indicating whether to create a full dump from the last entry in the dump table

create_incremental

```
./develop.sh manage dump create_incremental [OPTIONS]
```

Options

-l, --location <location>

-t, --threads <threads>

--dump-id <dump_id>

create_parquet

```
./develop.sh manage dump create_parquet [OPTIONS]
```

delete_old_dumps

```
./develop.sh manage dump delete_old_dumps [OPTIONS] LOCATION
```

Arguments

LOCATION
Required argument

import_dump

Import a ListenBrainz dump into the database.

Args: private_archive (str): the path to the ListenBrainz private dump to be imported private_timescale_archive (str): the path to the ListenBrainz private timescale dump to be imported public_archive (str): the path to the ListenBrainz public dump to be imported public_timescale_archive (str): the path to the ListenBrainz public timescale dump to be imported listen_archive (str): the path to the ListenBrainz listen dump archive to be imported threads (int): the number of threads to use during decompression, defaults to 1

Note: This method tries to import the private db dump first, followed by the public db dump. However, in absence of a private dump, it imports sanitized versions of the user table in the public dump in order to satisfy foreign key constraints. Then it imports the listen dump.

```
./develop.sh manage dump import_dump [OPTIONS]
```

Options

-pr, --private-archive <private_archive>
the path to the ListenBrainz private dump to be imported

--private-timescale-archive <private_timescale_archive>
the path to the ListenBrainz private timescale dump to be imported

-pu, --public-archive <public_archive>
the path to the ListenBrainz public dump to be imported

--public-timescale-archive <public_timescale_archive>
the path to the ListenBrainz public timescale dump to be imported

-l, --listen-archive <listen_archive>
Required the path to the ListenBrainz listen dump archive to be imported

-t, --threads <threads>
the number of threads to use during decompression, defaults to 1

import_yim_playlists

Import playlist excerpts into the YIM data table from a dump file.

Note: First copy the dump to inside the container from which the script is to be run.

Args: patch_slug (str): The slug of the troi patch that generated these playlists. dump_file (str): The dump file to import. For each user, it should contain three lines: user_name, playlist_mbid, JSPF data.

```
./develop.sh manage dump import_yim_playlists [OPTIONS] PATCH_SLUG DUMP_FILE
```

Arguments

PATCH_SLUG
Required argument

DUMP_FILE
Required argument

1.10.3 ListenBrainz Spark

These commands are used to interact with the Spark Cluster.

python spark_manage.py

```
python spark_manage.py [OPTIONS] COMMAND [ARGS]...
```

request_consumer

Invoke script responsible for the request consumer

```
python spark_manage.py request_consumer [OPTIONS]
```

./develop.sh manage spark

```
./develop.sh manage spark [OPTIONS] COMMAND [ARGS]...
```

cron_request_all_stats

```
./develop.sh manage spark cron_request_all_stats [OPTIONS]
```

cron_request_recommendations

```
./develop.sh manage spark cron_request_recommendations [OPTIONS]
```

cron_request_similar_users

```
./develop.sh manage spark cron_request_similar_users [OPTIONS]
```

request_candidate_sets

Send the cluster a request to generate candidate sets.

```
./develop.sh manage spark request_candidate_sets [OPTIONS]
```

Options

--days <days>
Request recommendations to be generated on history of given number of days

--top <top>
Calculate given number of top artist.

--similar <similar>
Calculate given number of similar artist.

--html
Enable/disable HTML file generation

--user-name <users>
Generate candidate set for given users. Generate for all active users by default.

request_dataframes

Send the cluster a request to create dataframes.

```
./develop.sh manage spark request_dataframes [OPTIONS]
```

Options

--days <days>
Request model to be trained on data of given number of days

--job-type <job_type>
The type of dataframes to request. 'recommendation_recording' or 'similar_users' are allowed.

--listens-threshold <listens_threshold>
The minimum number of listens a user should have to be included in the dataframes.

request_day_of_week

Send request to calculate most listened day of week to the spark cluster

```
./develop.sh manage spark request_day_of_week [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_import_artist_relation

Send the spark cluster a request to import artist relation.

```
./develop.sh manage spark request_import_artist_relation [OPTIONS]
```

request_import_full

Send the cluster a request to import a new full data dump

```
./develop.sh manage spark request_import_full [OPTIONS]
```

Options

--id <id_>
Optional. ID of the full dump to import, defaults to latest dump available on FTP server

request_import_incremental

Send the cluster a request to import a new incremental data dump

```
./develop.sh manage spark request_import_incremental [OPTIONS]
```

Options

--id <id_>
Optional. ID of the incremental dump to import, defaults to latest dump available on FTP server

request_import_musicbrainz_release_dump

Send the spark cluster a request to import musicbrainz release dump.

```
./develop.sh manage spark request_import_musicbrainz_release_dump
    [OPTIONS]
```

request_listens_per_day

Send request to calculate listens per day stat to the spark cluster

```
./develop.sh manage spark request_listens_per_day [OPTIONS]
```

request_model

Send the cluster a request to train the model.

For more details refer to https://spark.apache.org/docs/2.1.0/mllib-collaborative-filtering.html

```
./develop.sh manage spark request_model [OPTIONS]
```

Options

--rank <rank>
Number of hidden features

--itr <itr>
Number of iterations to run.

--lmbda <lmbda>
Controls over fitting.

--alpha <alpha>
Baseline level of confidence weighting applied.

request_most_listened_year

Send request to calculate most listened year stat to the spark cluster

```
./develop.sh manage spark request_most_listened_year [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_most_prominent_color

Send request to calculate most prominent color stat to the spark cluster

```
./develop.sh manage spark request_most_prominent_color [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_new_release_stats

Send request to calculate new release stats to the spark cluster

```
./develop.sh manage spark request_new_release_stats [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_recommendations

Send the cluster a request to generate recommendations.

```
./develop.sh manage spark request_recommendations [OPTIONS]
```

Options

--top <top>
Generate given number of top artist recommendations

--similar <similar>
Generate given number of similar artist recommendations

--user-name <users>
Generate recommendations for given users. Generate recommendations for all users by default.

request_similar_users

Send the cluster a request to generate similar users.

```
./develop.sh manage spark request_similar_users [OPTIONS]
```

Options

--max-num-users <max_num_users>
The maxiumum number of similar users to return for any given user.

request_similar_users_year_end

Send the cluster a request to generate similar users for Year in Music.

```
./develop.sh manage spark request_similar_users_year_end [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_sitewide_stats

Send request to calculate sitewide stats to the spark cluster

```
./develop.sh manage spark request_sitewide_stats [OPTIONS]
```

Options

--type <type_>
Required Type of statistics to calculate

Options entity | listening_activity

--range <range_>
Required Time range of statistics to calculate

Options week | month | quarter | half_yearly | year | all_time | this_week | this_month | this_year

--entity <entity>
Entity for which statistics should be calculated

Options artists | releases | recordings

request_user_stats

Send a user stats request to the spark cluster

```
./develop.sh manage spark request_user_stats [OPTIONS]
```

Options

--type <type_>
Required Type of statistics to calculate

Options entity | listening_activity | daily_activity

--range <range_>
Required Time range of statistics to calculate

Options week | month | quarter | half_yearly | year | all_time | this_week | this_month | this_year

--entity <entity>
Entity for which statistics should be calculated

Options artists | releases | recordings

request_year_in_music

```
./develop.sh manage spark request_year_in_music [OPTIONS]
```

request_yearly_listen_count

Send request to calculate yearly listen count stat to the spark cluster

```
./develop.sh manage spark request_yearly_listen_count [OPTIONS]
```

Options

--year <year>
Year for which to calculate the stat

request_yim_top_stats

Send request to calculate top stats to the spark cluster

```
./develop.sh manage spark request_yim_top_stats [OPTIONS]
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

/1

INDEX

Symbols

A

D

L

M

P

R

W