ListenBrainz Documentation

Release 0.1.0

MetaBrainz Foundation

Dec 30, 2025

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

All endpoints have this root URL for our current production site.

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

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

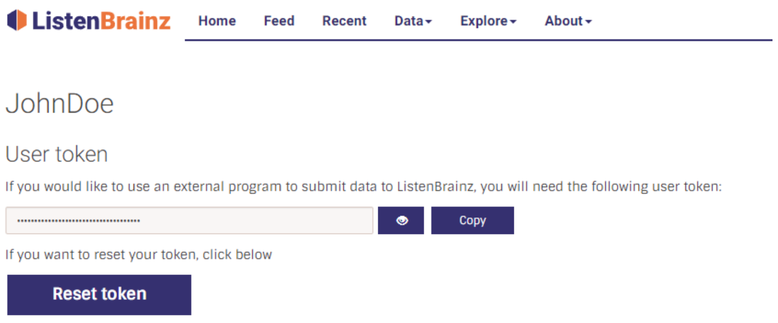
1.1.1 Authentication

ListenBrainz makes use of private API keys called user tokens to authenticate requests and ensure the proper access controls on user data. A user token is a unique alphanumeric string linked to a user account. To retrieve your user token, follow this guide.

Get the User token

Every account has a User token associated with it, to get the token:

1. Sign up or Log in your an account using this link.
2. Navigate to settings page to find your user Token (See image below for reference).



JohnDoe

User token

If you would like to use an external program to submit data to ListenBrainz, you will need the following user token:

If you want to reset your token, click below

3. Copy the User Token to your clipboard.

Note: You may also reset your user token by clicking the Reset token button on the settings page.

Add the User token to your requests

The user token must be included in the request header for its usage. To format the header correctly, you can use the following piece of code:

```
# 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)
}
```

Then include the formatted header in the request to use it.

```
  response = requests.get(
    ...
    # Your request url and params go here.
    ...
    headers=AUTH_HEADER,
)
```

Note: A complete usage example for a request employing Authorization headers to make authenticated requests to ListenBrainz can be found on the *API Usage* page.

1.1.2 Reference

Core

The ListenBrainz server supports the following end-points for submitting and fetching listens.

`GET /1/search/users/`

Search a ListenBrainz-registered user.

Parameters

- **search_term** – Input on which search operation is to be performed.

`POST /1/submit-listens`

Submit listens to the server. A user token (found on https://listenbrainz.org/settings/) 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*

GET /1/user/(mb_username: *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/user/(mb_username: *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/(mb_username: *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/(mb_username: 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/(mb_username: user_name)/similar-to/mb_username: 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/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.

Note: The listen is not deleted immediately, but is scheduled for deletion, which usually happens shortly after the hour.

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`

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/`(*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/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/recommendations**

Fetch recommendation playlist metadata in JSPF format without recordings for playlist_user_name. This endpoint only lists playlists that are to be shown on the listenbrainz.org recommendations pages.

Status Codes

- 200 OK – success
- 404 Not Found – user not found

Response Headers

- Content-Type – *application/json*

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

Search for a playlist by name for a user.

Parameters

- **playlist_user_name** – the MusicBrainz ID of the user whose playlists are being searched.

Query Parameters

- **name** – the name of the playlist to search for.
- **count** – the number of playlists to return. Default: 25.
- **offset** – the offset of the playlists to return. Default: 0.

Status Codes

- 200 OK – success
- 404 Not Found – user not found

Response Headers

- Content-Type – *application/json*

GET /1/user/(mb_username: *user_name*)**/services**

Get list of services which are connected to a given user's account.

```
{
    "user_name": "hwnrwx",
    "services": ["spotify"]
}
```

Parameters

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

Response Headers

- Content-Type – *application/json*

Status Codes

- 200 OK – Yay, you have data!
- 401 Unauthorized – Invalid authorization. See error message for details.
- 403 Forbidden – Forbidden, you do not have permissions to view this user's information.
- 404 Not Found – The requested user was not found.

`GET /1/lb-radio/tags`

Get recordings for use in LB radio with the specified tags that match the requested criteria.

Parameters

- **`tag`** – the MusicBrainz tag to fetch recordings for, this parameter can be specified multiple times. if more than one tag is specified, the operator param should also be specified.
- **`operator`** – specify AND to retrieve recordings that have all the tags, otherwise specify OR to retrieve recordings that have any one of the tags.
- **`pop_begin`** – percent is a measure of the recording's popularity, pop_begin denotes a preferred lower bound on the popularity of recordings to be returned.
- **`pop_end`** – percent is a measure of the recording's popularity, pop_end denotes a preferred upper bound on the popularity of recordings to be returned.
- **`count`** – number of recordings to return for the

Response Headers

- Content-Type – *application/json*

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – Invalid or missing param in request, see error message for details.

`GET /1/lb-radio/artist/`(*seed_artist_mbid*)

Get recordings for use in LB radio with the given seed artist. The endpoint returns a dict of all the similar artists, including the seed artist. For each artists, there will be a list of dicts that contain recording_mbid, similar_artist_mbid and total_listen_count:

```
{
  "recording_mbid": "401c1a5d-56e7-434d-b07e-a14d4e7eb83c",
  "similar_artist_mbid": "cb67438a-7f50-4f2b-a6f1-2bb2729fd538",
  "similar_artist_name": "Boo Hoo Boys",
  "total_listen_count": 232361
}
```

Parameters

- **`mode`** – mode is the LB radio mode to be used for this query. Must be one of "easy", "medium", "hard".
- **`max_similar_artists`** – The maximum number of similar artists to return recordings for.
- **`max_recordings_per_artist`** – The maximum number of recordings to return for each artist. If there are aren't enough recordings, all available recordings will be returned.
- **`pop_begin`** – Popularity range percentage lower bound. A popularity range is given to narrow down the recordings into a smaller target group. The most popular recording(s) on LB have a pop percent of 100. The least popular recordings have a score of 0. This range is not coupled to the specified mode, but the mode would often determine the popularity range, so that less popular recordings can be returned on the medium and harder modes.
- **`pop_end`** – Popularity range percentage upper bound. See above.

Response Headers

- Content-Type – *application/json*

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – Invalid or missing param in request, see error message for details.

`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/settings/.

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.

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_PAYLOAD_SIZE = 10240000`**

The maximum size of a payload in bytes. The same as MAX_LISTEN_SIZE * MAX_LISTENS_PER_REQUEST.

`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_DURATION_LIMIT = 2073600`**

The max permitted value of duration field - 24 days

`listenbrainz.webserver.views.api_tools.`**`MAX_DURATION_MS_LIMIT = 2073600000`**

The max permitted value of duration_ms field - 24 days

`listenbrainz.webserver.views.api_tools.`**`MAX_LISTENS_PER_REQUEST = 1000`**

The maximum number of listens in a request.

`listenbrainz.webserver.views.api_tools.`**`MAX_ITEMS_PER_GET = 1000`**

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

Playlists

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

`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/`(*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/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*

`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*

`GET /1/playlist/search`

Search for playlists by name or description. The search query must be at least 3 characters long.

Parameters

- **`query`** (`str`) – The search query string.

Status Codes

- 200 OK – Yay, you have data!
- 400 Bad Request – invalid query string, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

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*

`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*

GET /1/playlist/(*playlist_mbid*)**/xspf**

Fetch the given playlist as XSPF.

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/xspf+xml*

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/(*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,
```

(continues on next page)

(continued from previous page)

```
    "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/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*)/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*

`POST /1/playlist/`(*playlist_mbid*)`/export/`
service

Export a playlist to an external service, given a playlist MBID.

Request Headers

- Authorization – Token <user token>

Parameters

- **`playlist_mbid`** – The playlist mbid to export.
- **`is_public`** – Should the exported playlist be public or not?

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/import/`(*service*)

Get playlists from chosen Music Service. :reqheader Authorization: Token <user token> :statuscode 200: playlists are fetched. :statuscode 401: invalid authorization. See error message for details. :statuscode 404: Playlists not found :resheader Content-Type: *application/json*

`GET /1/playlist/spotify/`(*playlist_id*)`/tracks`

Import a playlist's tracks from Spotify and convert them to JSPF.

Request Headers

- Authorization – Token <user token>

Parameters

- **`playlist_id`** – The Spotify playlist id to get the tracks from

Status Codes

- 200 OK – tracks are fetched and converted.
- 401 Unauthorized – invalid authorization. See error message for details.
- 404 Not Found – Playlist not found

Response Headers

- Content-Type – *application/json*

`GET /1/playlist/apple_music/`(*playlist_id*)**`/tracks`**

Import a playlist's tracks from Apple Music and convert them to JSPF.

Request Headers

- Authorization – Token <user token>

Parameters

- **`playlist_id`** – The Apple Music playlist id to get the tracks from

Status Codes

- 200 OK – tracks are fetched and converted.
- 401 Unauthorized – invalid authorization. See error message for details.
- 404 Not Found – Playlist not found

Response Headers

- Content-Type – *application/json*

`GET /1/playlist/soundcloud/`(*playlist_id*)**`/tracks`**

Import a playlist tracks from a SoundCloud and convert them to JSPF. :reqheader Authorization: Token <user token> :param playlist_id: The SoundCloud playlist id to get the tracks from :statuscode 200: tracks are fetched and converted. :statuscode 401: invalid authorization. See error message for details. :statuscode 404: Playlist not found :resheader Content-Type: *application/json*

`POST /1/playlist/export-jspf/`(*service*)

Export a playlist to an external service from JSPF POSTed to this endpoint.

Request Headers

- Authorization – Token <user token>

Parameters

- **`is_public`** – Should the exported playlist be public or not?

Status Codes

- 200 OK – playlist copied.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

Recordings

Feedback API

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/settings/) 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/user/(mb_username:` *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*

`GET /1/feedback/recording/(`*recording_mbid*`)/get-feedback-mbid`

Get feedback for recording with given `recording_mbid`. 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/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*

POST /1/feedback/user/(mb_username: *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.

The format of the post data should look as follows:

```
{
    "recording_msids": ["<msid1>", "<msid2>", "<msid3>"],
    "recording_mbids": ["<mbid1>", "<mbid2>", "<mbid3>"]
}
```

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

GET /1/feedback/user/(mb_username: *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.

Note: If you get a 502 error while querying this endpoint using a GET request, consider reducing the number of total recordings you are querying in 1 request. As a rule of thumb, requesting maximum ~75 recordings in 1 request will avert the error.

The reason this error occurs is because the recording uuids are query params which are part of the request url. The length of the url is subject to a general limit imposed at the middleware level so requests with long urls never reach the ListenBrainz backend. Due to the same reason, the backend cannot provide a meaningful error.

Parameters

- **recordings** (str) – comma separated list of recording_msids for which feedback records are to be fetched. this param is deprecated and will be removed in the future. use recording_msids instead.
- **recording_msids** (str) – comma separated list of recording_msids for which feedback records are to be fetched.
- **recording_mbids** (str) – comma separated list of recording_mbids for which feedback records are to be fetched.

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

POST /1/feedback/import

Import feedback from external service.

Pinned Recording API

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

POST /1/pin

Pin a recording for user. A user token (found on https://listenbrainz.org/settings/) 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/unpin`

Unpins the currently active pinned recording for the user. A user token (found on https://listenbrainz.org/settings/) 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/delete/`(*row_id*)

Deletes the pinned recording with given `row_id` from the server. A user token (found on https://listenbrainz.org/settings/) 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/(mb_username:` *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"
```

(continues on next page)

(continued from previous page)

```
            }
        },
        "-- 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*

GET /1/(mb_username: *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 ---"
    ],
```

(continues on next page)

(continued from previous page)

```
    "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/(mb_username: *user_name*)/pins/current

Get the currently pinned recording by a user with given `user_name`. The JSON returned by the API will look like the following:

```
{
    "pinned_recording": {
        "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"
        }
    },
    "user_name": "-- the MusicBrainz ID of the user --"
}
```

If there is no current pin for the user, "pinned_recording" field will be null.

Parameters

- **user_name** (str) – the MusicBrainz ID of the user whose pin track history requested.

Status Codes

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

Response Headers

- Content-Type – *application/json*

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

Updates the blurb content of a pinned recording for the user. A user token (found on https://listenbrainz.org/settings/) 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:

```
{
    "blurb_content": "Wow..",
}
```

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*

Statistics

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/user/(**mb_username:** *user_name*)**/artists**

Get top artists for user `user_name`.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "artists": [
            {
               "artist_mbid": "93e6118e-7fa8-49f6-9e02-699a1ebce105",
               "artist_name": "The Local train",
               "listen_count": 385
            },
            {
               "artist_mbid": "ae9ed5e2-4caf-4b3d-9cb3-2ad626b91714",
               "artist_name": "Lenka",
               "listen_count": 333
            },
            {
               "artist_mbid": "cc197bad-dc9c-440d-a5b5-d52ba2e14234",
               "artist_name": "Coldplay",
               "listen_count": 321
            }
```

(continues on next page)

(continued from previous page)

```
        ],
        "count": 3,
        "total_artist_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note: `artist_mbid` is an optional field 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*

GET /1/stats/user/(mb_username: *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"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "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: `artist_mbids` and `release_mbid` 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/(mb_username: *user_name***)/release-groups**

Get top release groups for user `user_name`.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "release_groups": [
            {
                "artist_mbids": [
                    "62162215-b023-4f0e-84bd-1e9412d5b32c",
                    "faf4cefb-036c-4c88-b93a-5b03dd0a0e6b",
                    "e07d9474-00ea-4460-ac27-88b46b3d976e"
                ],
                "artist_name": "All Time Low ft. Demi Lovato & blackbear",
                "caa_id": 29179588350,
                "caa_release_mbid": "ee65192d-31f3-437a-b170-9158d2172dbc",
                "listen_count": 456,
                "release_group_mbid": "326b4a29-dff5-4fab-87dc-efc1494001c6",
                "release_group_name": "Monsters"
            },
            {
                "artist_mbids": [
                    "c8b03190-306c-4120-bb0b-6f2ebfc06ea9"
                ],
                "artist_name": "The Weeknd",
                "caa_id": 25720993837,
                "caa_release_mbid": "19e4f6cc-ca0c-4897-8dfc-a36914b7f998",
                "listen_count": 381,
                "release_group_mbid": "78570bea-2a26-467c-a3db-c52723ceb394",
                "release_group_name": "After Hours"
            }
        ],
        "count": 2,
        "total_release_group_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note: `artist_mbids` and `release_group_mbid` 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/(mb_username: *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",
                "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:

- We only calculate the top 1000 all_time recordings
- `artist_mbids`, `release_name`, `release_mbid` and `recording_mbid` 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/(mb_username: *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"
    }
}
```

Note:

- 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/(mb_username: *user_name*)**/artist-activity**

Get the artist activity for user `user_name`. The artist activity shows the total number of listens for each artist along with their albums and corresponding listen counts.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "artist_activity": [
            {
                "name": "Radiohead",
                "artist_name": "Radiohead",
                "artist_mbid": "a74b1b7f-71a5-4011-9441-dc7410c7388a",
                "listen_count": 120,
                "albums": [
                    {
                        "name": "OK Computer",
                        "listen_count": 45,
                        "release_group_mbid": "1c2b57e1-9b3d-4f5a-8c7d-9e0f1a2b3c4d"
                    },
                    {
                        "name": "In Rainbows",
                        "listen_count": 75,
                        "release_group_mbid": null
                    }
                ]
            },
            {
                "name": "The Beatles",
```

(continues on next page)

(continued from previous page)

```
                "artist_name": "The Beatles",
                "artist_mbid": null,
                "listen_count": 95,
                "albums": [
                    {
                        "name": "Abbey Road",
                        "listen_count": 60,
                        "release_group_mbid": "2d3c68f2-0a4e-5g6b-9d8e-0f1a2b3c4d5e"
                    },
                    {
                        "name": "Revolver",
                        "listen_count": 35,
                        "release_group_mbid": null
                    }
                ]
            }
        ],
        "user_id": "foobar",
        "range": "all_time",
        "from_ts": 1609459200,
        "to_ts": 1640995200,
        "last_updated": 1640995200
    }
}
```

Note:

- The example above shows artist activity data with two artists and their respective albums.
- The statistics are aggregated based on the number of listens recorded for each artist and their albums.
- **Each artist entry includes:**
  - `name`: The artist name (may be the credited name on the release group)
  - `artist_name`: The canonical artist name (only present for user-specific requests when available)
  - `artist_mbid`: The MusicBrainz artist ID (may be null if unavailable)
  - `listen_count`: Total number of listens for this artist
  - `albums`: List of albums/release groups for this artist
- **Each album entry includes:**
  - `name`: The release group name
  - `listen_count`: Number of listens for this release group
  - `release_group_mbid`: The MusicBrainz release group ID (may be null if unavailable)

Parameters

- **range** (`str`) – Optional stats range (see *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/(mb_username: ` *user_name*`)/era-activity`

Get the release-year activity for user `user_name`. Each entry represents the number of listens to recordings whose **original release year** equals the listed `year`. (Frontends may group these years into decades to present a classic "era" visualization.)

A sample response from the endpoint may look like:

```
{
    "payload": {
        "era_activity": [
            {"year": 1971, "listen_count": 3},
            {"year": 1997, "listen_count": 9},
            {"year": 2024, "listen_count": 1}
        ],
        "from_ts": 315532800,
        "to_ts": 1735603200,
        "range": "week",
        "last_updated": 1735603200,
        "user_id": "John Doe"
    }
}
```

Note:

- `year` is the recording's release year; multiple listens to different tracks from the same year are aggregated.
- Clients may bucket by decade (e.g. 1970s, 1990s) if they want true "era" bars.
- Empty years are omitted (only years with > 0 listens are returned for the selected range).

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/(mb_username: *user_name***)/genre-activity**

Get the genre activity for user `user_name`. The genre activity shows the total number of listens for each genre broken down by hour of the day.

A sample response from the endpoint may look like:

```
{
    "result": [
        {
            "genre": "alternative dance",
            "hour": 14,
            "listen_count": 3
        },
        {
            "genre": "alternative punk",
            "hour": 11,
            "listen_count": 6
        },
        {
            "genre": "alternative rock",
            "hour": 11,
            "listen_count": 8
        },
        {
            "genre": "electronic",
            "hour": 10,
            "listen_count": 13
        },
        {
            "genre": "rock",
            "hour": 11,
            "listen_count": 16
        }
    ]
}
```

Note:

- The example above shows genre activity data with listening patterns across different hours.
- Each entry represents the number of times a genre was listened to during a specific hour of the day.
- Hours are in 24-hour format (0-23).
- The statistics help identify when users prefer to listen to different genres throughout the day.

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/(mb_username: *user_name*)**/artist-evolution-activity**

Get the artist evolution activity for a specific user. Over the selected time range, this returns raw rows of listen counts per artist per time unit (e.g., weekday, day-of-month, month, or year). The structure mirrors the sitewide endpoint.

A sample response may look like:

```
{
  "payload": {
    "artist_evolution_activity": [
      { "time_unit": "Monday",    "artist_mbid": "mbid_taylor",  "artist_name":
↪"Taylor Swift", "listen_count": 120 },
      { "time_unit": "Monday",    "artist_mbid": "mbid_drake",   "artist_name":
↪"Drake",        "listen_count": 80  },
      { "time_unit": "Sunday",    "artist_mbid": "mbid_weeknd",  "artist_name":
↪"The Weeknd",   "listen_count": 400 }
    ],
    "range": "week",
    "from_ts": 1609459200,
    "to_ts": 1640995200,
    "last_updated": 1640995200,
    "user_id": "foobar"
  }
}
```

Note:

- **time_unit depends on the stats range:**
  - week → weekday names (Monday..Sunday)
  - month → day numbers as strings (“1”..”31”)
  - year → month names (January..December)
  - all_time → calendar years as strings (“2019”, “2020”, …)
- artist_mbid may be null/omitted if unavailable.

Parameters

- **range** (str) – Optional stats range (see *ALLOWED_STATISTICS_RANGE*), defaults to all_time.

Status Codes

- 200 OK – Successful query.
- 204 No Content – Statistics not available.
- 400 Bad Request – Bad request.
- 404 Not Found – User not found.

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(mb_username: *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,
                },
                "..."
            ],
            "Tuesday": ["..."],
            "..."
        },
        "stats_range": "all_time",
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
    }
}
```

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/(mb_username: *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"
    }
}
```

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/artist/(*artist_mbid*)**/listeners**

Get top listeners for artist `artist_mbid`. This includes the total listen count for the entity and top N listeners with their individual listen count for that artist in a given time range. A sample response from the endpoint may look like:

```
{
  "payload": {
```

(continues on next page)

(continued from previous page)

```
    "artist_mbid": "00034ede-a1f1-4219-be39-02f36853373e",
    "artist_name": "O Rappa",
    "from_ts": 1009843200,
    "last_updated": 1681839677,
    "listeners": [
      {
        "listen_count": 2469,
        "user_name": "RosyPsanda"
      },
      {
        "listen_count": 1858,
        "user_name": "alexyagui"
      },
      {
        "listen_count": 578,
        "user_name": "rafael_gn"
      },
      {
        "listen_count": 8,
        "user_name": "italooliveira"
      },
      {
        "listen_count": 7,
        "user_name": "paulodesouza"
      },
      {
        "listen_count": 1,
        "user_name": "oldpunisher"
      }
    ],
    "stats_range": "all_time",
    "to_ts": 1681777035,
    "total_listen_count": 16393
  }
}
```

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 or the entity does not exist, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – Entity not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/release-group/(*release_group_mbid*)**/listeners**

Get top listeners for release group `release_group_mbid`. This includes the total listen count for the entity and top N listeners with their individual listen count for that release group in a given time range. A sample response from the endpoint may look like:

```
{
  "payload": {
    "artist_mbids": [
      "c234fa42-e6a6-443e-937e-2f4b073538a3"
    ],
    "artist_name": "Chris Brown",
    "caa_id": 23564822587,
    "caa_release_mbid": "25f18616-5a9c-470e-964d-4eb8a511435b",
    "from_ts": 1009843200,
    "last_updated": 1681843150,
    "listeners": [
      {
        "listen_count": 2365,
        "user_name": "purpleyor"
      },
      {
        "listen_count": 570,
        "user_name": "dndty"
      },
      {
        "listen_count": 216,
        "user_name": "iammsyre"
      },
      {
        "listen_count": 141,
        "user_name": "dpmittal"
      },
      {
        "listen_count": 33,
        "user_name": "tazlad"
      },
      {
        "listen_count": 30,
        "user_name": "ratkutti"
      },
      {
        "listen_count": 22,
        "user_name": "Raymorjamiek"
      },
      {
        "listen_count": 21,
        "user_name": "MJJMC"
      },
      {
        "listen_count": 12,
        "user_name": "fookever"
      },
      {
```

(continues on next page)

(continued from previous page)

```
        "listen_count": 8,
        "user_name": "Jamjamk12071983"
      },
      {
        "listen_count": 1,
        "user_name": "hassanymoses"
      },
      {
        "listen_count": 1,
        "user_name": "iJays"
      }
    ],
    "release_group_mbid": "087b3a7d-d532-44d9-b37a-84427677ddcd",
    "release_group_name": "Indigo",
    "stats_range": "all_time",
    "to_ts": 1681777035,
    "total_listen_count": 10291
  }
}
```

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 or the entity does not exist, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – Entity not found

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_mbid": null,
                "artist_name": "Kanye West",
                "listen_count": 1305
            },
            {
                "artist_mbid": "0b30341b-b59d-4979-8130-b66c0e475321",
                "artist_name": "Lil Nas X",
```

(continues on next page)

(continued from previous page)

```
                "listen_count": 1267
            }
        ],
        "offset": 0,
        "count": 2,
        "total_artist_count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note:

- `artist_mbid` is optional field 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/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"
```

(continues on next page)

(continued from previous page)

```
            },
            {
                "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,
        "total_release_count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note:

- `artist_mbids` and `release_mbid` 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/release-groups`

Get sitewide top release groups.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "release_groups": [
            {
                "artist_mbids": [
                    "62162215-b023-4f0e-84bd-1e9412d5b32c",
                    "faf4cefb-036c-4c88-b93a-5b03dd0a0e6b",
                    "e07d9474-00ea-4460-ac27-88b46b3d976e"
                ],
                "artist_name": "All Time Low ft. Demi Lovato & blackbear",
                "caa_id": 29179588350,
                "caa_release_mbid": "ee65192d-31f3-437a-b170-9158d2172dbc",
                "listen_count": 456,
                "release_group_mbid": "326b4a29-dff5-4fab-87dc-efc1494001c6",
                "release_group_name": "Monsters"
            },
            {
                "artist_mbids": [
                    "c8b03190-306c-4120-bb0b-6f2ebfc06ea9"
                ],
                "artist_name": "The Weeknd",
                "caa_id": 25720993837,
                "caa_release_mbid": "19e4f6cc-ca0c-4897-8dfc-a36914b7f998",
                "listen_count": 381,
                "release_group_mbid": "78570bea-2a26-467c-a3db-c52723ceb394",
                "release_group_name": "After Hours"
            }
        ],
        "offset": 0,
        "count": 2,
        "total_release_group_count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note:

- `artist_mbids` and `release_mbid` 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/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,
        "total_recording_count": 2,
        "range": "year",
        "last_updated": 1588494361,
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```

Note:

- We only calculate the top 1000 all_time recordings
- `artist_mbids`, `release_name`, `release_mbid` and `recording_mbid` 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/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:

- 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/artist-activity`

Get the sitewide artist activity. The artist activity shows the total number of listens across all users for each artist along with their albums and corresponding listen counts.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "artist_activity": [
            {
                "name": "Radiohead",
                "artist_mbid": null,
                "listen_count": 12000,
                "albums": [
                    {
                        "name": "OK Computer",
                        "listen_count": 4500,
                        "release_group_mbid": "1c2b57e1-9b3d-4f5a-8c7d-9e0f1a2b3c4d"
                    },
                    {
                        "name": "In Rainbows",
                        "listen_count": 7500,
                        "release_group_mbid": null
                    }
                ]
            },
            {
                "name": "The Beatles",
                "artist_mbid": null,
                "listen_count": 9500,
                "albums": [
                    {
```

(continues on next page)

(continued from previous page)

```
                        "name": "Abbey Road",
                        "listen_count": 6000,
                        "release_group_mbid": "2d3c68f2-0a4e-5g6b-9d8e-0f1a2b3c4d5e"
                    },
                    {
                        "name": "Revolver",
                        "listen_count": 3500,
                        "release_group_mbid": null
                    }
                ]
            }
        ],
        "range": "all_time",
        "from_ts": 1587945600,
        "to_ts": 1640995200,
        "last_updated": 1640995200
    }
}
```

Note:

- The example above shows sitewide artist activity data with two artists and their respective albums.
- The statistics are aggregated based on the number of listens recorded across all users for each artist and their albums.
- **Each artist entry includes:**
  - `name`: The artist name
  - `artist_mbid`: The MusicBrainz artist ID (always null for sitewide stats)
  - `listen_count`: Total number of listens for this artist across all users
  - `albums`: List of albums/release groups for this artist
- **Each album entry includes:**
  - `name`: The release group name
  - `listen_count`: Number of listens for this release group across all users
  - `release_group_mbid`: The MusicBrainz release group ID (may be null if unavailable)

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 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/era-activity`

Get sitewide release-year activity. Each entry represents the number of listens across all users to recordings whose **original release year** equals the listed `year`. (Frontends may group these years into decades to present a classic "era" visualization.)

A sample response from the endpoint may look like:

```
{
    "payload": {
        "era_activity": [
            {"year": 1973, "listen_count": 1043},
            {"year": 1997, "listen_count": 3877},
            {"year": 2011, "listen_count": 2610}
        ],
        "from_ts": 315532800,
        "to_ts": 1735603200,
        "range": "year",
        "last_updated": 1735603200
    }
}
```

Note:

- `year` is the recording's release year; counts are aggregated across the entire ListenBrainz userbase.
- Clients may bucket by decade (e.g. 1970s, 1990s) if they want true "era" bars.
- Empty years are omitted.

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 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/artist-evolution-activity`

Get the sitewide artist evolution activity. Over the selected time range, this returns raw rows of listen counts per artist per time unit (e.g., weekday, day-of-month, month, or year). The structure mirrors the user endpoint.

A sample response may look like:

```
{
  "payload": {
    "artist_evolution_activity": [
      { "time_unit": "Monday",    "artist_mbid": "mbid_taylor",   "artist_name":
↪"Taylor Swift", "listen_count": 120 },
```

(continues on next page)

(continued from previous page)

```
      { "time_unit": "Tuesday",   "artist_mbid": "mbid_drake",   "artist_name":
↪"Drake",        "listen_count": 200 },
      { "time_unit": "Sunday",    "artist_mbid": "mbid_weeknd",  "artist_name":
↪"The Weeknd",   "listen_count": 400 }
    ],
    "range": "week",
    "from_ts": 1609459200,
    "to_ts": 1640995200,
    "last_updated": 1640995200
  }
}
```

Note:

- **`time_unit` depends on the stats range:**
  - `week` → weekday names (Monday..Sunday)
  - `month` → day numbers as strings ("1".."31")
  - `year` → month names (January..December)
  - `all_time` → calendar years as strings ("2019", "2020", …)
- `artist_mbid` may be null/omitted if unavailable.
- Shape matches `/user/<user_name>/artist-evolution-activity` for easy client reuse.

Parameters

- **range** (`str`) – Optional stats range (see *ALLOWED_STATISTICS_RANGE*), defaults to `all_time`.

Status Codes

- 200 OK – Successful query.
- 204 No Content – Statistics not available.
- 400 Bad Request – Bad request.

Response Headers

- Content-Type – *application/json*

GET /1/stats/sitewide/artist-map

Get the sitewide artist map. The artist map shows the number of artists listened to by users from different countries of the world.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "artist_map": [
            {
```

(continues on next page)

(continued from previous page)

```
                "country": "USA",
                "artist_count": 34
            },
            {
                "country": "GBR",
                "artist_count": 69
            },
            {
                "country": "IND",
                "artist_count": 32
            }
        ],
        "stats_range": "all_time"
        "to_ts": 1589155200,
    }
}
```

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/(mb_username: *user_name*)**/year-in-music/legacy/**
int: *year*

Get data for legacy year in music stuff

GET /1/stats/user/(mb_username: *user_name*)**/year-in-music/**
int: *year*

GET /1/stats/user/(mb_username: *user_name*)**/year-in-music**

Get data for year in music stuff

Constants

Constants that are relevant to using the API:

`data.model.common_stat.ALLOWED_STATISTICS_RANGE = ['this_week', 'this_month', 'this_year', 'week', 'month', 'quarter', 'year', 'half_yearly', 'all_time']`

list of allowed value for range param accepted by various statistics endpoints

Popularity

The popularity APIs return the total listen and listeners count for various entities and also a way to query top entities for a given artist.

`GET /1/popularity/top-recordings-for-artist/`(*artist_mbid*)

Get the top recordings by listen count for a given artist. The response is of the following format:

```
[
  {
    "artist_mbids": [
      "b7ffd2af-418f-4be2-bdd1-22f8b48613da"
    ],
    "artist_name": "Nine Inch Nails",
    "caa_id": 2546761764,
    "caa_release_mbid": "2d410836-5add-3661-b0b0-168ba1696611",
    "length": 373133,
    "recording_mbid": "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
    "recording_name": "Closer",
    "release_color": {
        "blue": 104,
        "green": 104,
        "red": 84
    },
    "release_mbid": "ba8701ba-dc7c-4bca-9c83-846ee8c3d576",
    "release_name": "The Downward Spiral",
    "total_listen_count": 1380798,
    "total_user_count": 129454
  }
]
```

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid artist_mbid

`GET /1/popularity/top-release-groups-for-artist/`(*artist_mbid*)

Get the top release groups by listen count for a given artist. The response is of the following format:

```
[
  {
    "artist": {
      "artist_credit_id": 368737,
      "artists": [],
```

(continues on next page)

(continued from previous page)

```
      "name": "Pritam"
    },
    "release": {
      "caa_id": 14996821464,
      "caa_release_mbid": "488ef20e-7a2b-4daf-8bee-4f54fe26c7ab",
      "date": "2016-10-26",
      "name": "Ae Dil Hai Mushkil",
      "rels": [],
      "type": "Album"
    },
    "release_color": {
      "blue": 64,
      "green": 69,
      "red": 113
    },
    "release_group": {
      "caa_id": 14996821464,
      "caa_release_mbid": "488ef20e-7a2b-4daf-8bee-4f54fe26c7ab",
      "date": "2016-10-26",
      "name": "Ae Dil Hai Mushkil",
      "rels": [],
      "type": "Album"
    },
    "release_group_mbid": "d0991cc9-2277-4f5e-bd4d-2fa44507f623",
    "tag": {},
    "total_listen_count": 1432,
    "total_user_count": 82
  },
]
```

Parameters

- **artist_mbid** (str) – the mbid of the artist to get top release groups for

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid artist_mbid

POST /1/popularity/recording

Get the total listen count and total unique listeners count for a given recording.

A JSON document with a list of recording_mbids and inc string must be POSTed. Up to `MAX_ITEMS_PER_GET` items can be requested at once. Example:

```
{
    "recording_mbids": [
        "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "22ad712e-ce73-9f0c-4e97-9f0e53144411"
    ]
}
```

The response maintains the order of the recording mbids supplied and also includes any recordings for which the data was not found with counts set to null. Example:

```
[
    {
        "recording_mbid": "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "total_listen_count": 1000,
        "total_user_count": 10
    },
    {
        "recording_mbid": "22ad712e-ce73-9f0c-4e97-9f0e53144411",
        "total_listen_count": null,
        "total_user_count": null
    }
]
```

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid recording_mbid(s)

`POST /1/popularity/artist`

Get the total listen count and total unique listeners count for a given artist.

A JSON document with a list of artists and inc string must be POSTed. Up to `MAX_ITEMS_PER_GET` items can be requested at once. Example:

```
{
    "artist_mbids": [
        "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "22ad712e-ce73-9f0c-4e97-9f0e53144411"
    ]
}
```

The response maintains the order of the artist mbids supplied and also includes any artists for which the data was not found with counts set to null. Example:

```
[
    {
        "artist_mbid": "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "total_listen_count": 1000,
        "total_user_count": 10
    },
    {
        "artist_mbid": "22ad712e-ce73-9f0c-4e97-9f0e53144411",
        "total_listen_count": null,
        "total_user_count": null
    }
]
```

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid artist_mbid(s)

`POST /1/popularity/release`

Get the total listen count and total unique listeners count for a given release.

A JSON document with a list of releases and inc string must be POSTed. Up to `MAX_ITEMS_PER_GET` items can be requested at once. Example:

```
{
    "release_mbids": [
        "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "22ad712e-ce73-9f0c-4e97-9f0e53144411"
    ]
}
```

The response maintains the order of the release mbids supplied and also includes any releases for which the data was not found with counts set to null. Example:

```
[
    {
        "release_mbid": "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "total_listen_count": 1000,
        "total_user_count": 10
    },
    {
        "release_mbid": "22ad712e-ce73-9f0c-4e97-9f0e53144411",
        "total_listen_count": null,
        "total_user_count": null
    }
]
```

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid release_mbid(s)

`POST /1/popularity/release-group`

Get the total listen count and total unique listeners count for a given release group.

A JSON document with a list of release groups and inc string must be POSTed. Up to `MAX_ITEMS_PER_GET` items can be requested at once. Example:

```
{
    "release_group_mbids": [
        "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "22ad712e-ce73-9f0c-4e97-9f0e53144411"
    ]
}
```

The response maintains the order of the release group mbids supplied and also includes any release groups for which the data was not found with counts set to null. Example:

```
[
    {
        "release_group_mbid": "13dd61c7-ce73-4e97-9f0c-9f0e53144411",
        "total_listen_count": 1000,
```

(continues on next page)

(continued from previous page)

```
        "total_user_count": 10
    },
    {
        "release_group_mbid": "22ad712e-ce73-9f0c-4e97-9f0e53144411",
        "total_listen_count": null,
        "total_user_count": null
    }
]
```

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid release_group_mbid(s)

Metadata

The metadata API looks up MusicBrainz metadata for recordings

`GET /1/metadata/recording/`

This endpoint takes in a list of recording_mbids and returns an array of dicts that contain recording metadata suitable for showing in a context that requires as much detail about a recording and the artist. Using the inc parameter, you can control which portions of metadata to fetch.

The data returned by this endpoint can be seen here:

```
{
   "e97f805a-ab48-4c52-855e-07049142113d": {
      "tag": {
         "recording": [
            {
               "count": 8,
               "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
               "tag": "trip hop"
            },
            {
               "count": 3,
               "genre_mbid": "89255676-1f14-4dd8-bbad-fca839d6aff4",
               "tag": "electronic"
            },
            {
               "count": 4,
               "tag": "trip-hop"
            }
         ],
         "artist": [
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 13,
               "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
               "tag": "trip hop"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
                "count": 7,
                "tag": "trip-hop"
            }
        ],
        "release_group": [
            {
                "count": 18,
                "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
                "tag": "trip hop"
            },
            {
                "count": 6,
                "tag": "trip-hop"
            },
            {
                "count": 1,
                "tag": "mysterious"
            }
        ]
    },
    "recording": {
        "rels": [
            {
                "artist_name": "Beth Gibbons",
                "instrument": null,
                "artist_mbid": "5adcb9d9-5ea2-428d-af46-ef626966e106",
                "type": "vocal"
            },
            {
                "artist_mbid": "5082a11f-7203-4ff3-ae04-2a0150d3bbb6",
                "type": "instrument",
                "instrument": "Rhodes piano",
                "artist_name": "Geoff Barrow"
            },
            {
                "type": "instrument",
                "artist_mbid": "619b1116-740e-42e0-bdfe-96af274f79f7",
                "instrument": "guitar",
                "artist_name": "Adrian Utley"
            },
            {
                "artist_name": "Clive Deamer",
                "instrument": "drums (drum set)",
                "type": "instrument",
                "artist_mbid": "d576e6be-03d1-489c-8c3e-692c6fbfb7ca"
            }
        ]
    },
    "release": {
        "album_artist_name": "Portishead",
        "caa_id": 829521842,
```

(continues on next page)

(continued from previous page)

```
                "caa_release_mbid": "76df3287-6cda-33eb-8e9a-044b5e15ffdd",
                "mbid": "76df3287-6cda-33eb-8e9a-044b5e15ffdd",
                "name": "Dummy",
                "release_group_mbid": "48140466-cff6-3222-bd55-63c27e43190d",
                "year": 1994
            },
            "artist": {
                "artist_credit_id": 65,
                "name": "Portishead",
                "artists": [
                    {
                        "area": "United Kingdom",
                        "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
                        "begin_year": 1991,
                        "join_phrase": "",
                        "name": "Portishead",
                        "rels": {
                            "free streaming": "https://www.deezer.com/artist/1069",
                            "lyrics": "https://muzikum.eu/en/122-6105/portishead/lyrics.html",
                            "official homepage": "http://www.portishead.co.uk/",
                            "purchase for download": "https://www.junodownload.com/artists/
↪Portishead/releases/",
                            "social network": "https://www.facebook.com/portishead",
                            "streaming": "https://tidal.com/artist/27441",
                            "wikidata": "https://www.wikidata.org/wiki/Q191352",
                            "youtube": "https://www.youtube.com/user/portishead1002"
                        },
                        "type": "Group"
                    }
                ]
            }
        }
}
```

Parameters

- **recording_mbids** (str) – A comma separated list of recording_mbids
- **inc** (str) – A space separated list of "artist", "tag" and/or "release" to indicate which portions of metadata you're interested in fetching. We encourage users to only fetch the data they plan to consume.

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid recording_mbid arguments

`POST /1/metadata/recording/`

This endpoint is the POST verson for fetching recording metadata, since it allows up to the max number of items allowed. (`MAX_ITEMS_PER_GET` items)

A JSON document with a list of recording_mbids and inc string must be POSTed to this endpoint to returns an array of dicts that contain recording metadata suitable for showing in a context that requires as much detail about a recording and the artist. Using the inc parameter, you can control which portions of metadata to fetch.

```
{
     "recording_mbids": [ "25d47b0c-5177-49db-b740-c166e4acebd1", "..." ],
     "inc": "artist tag"
}
```

To see what data this endpoint returns, please look at the data above for the GET version.

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid recording_mbid arguments

GET /1/metadata/release_group/

This endpoint takes in a list of release_group_mbids and returns an array of dicts that contain release_group metadata suitable for showing in a context that requires as much detail about a release_group and the artist. Using the inc parameter, you can control which portions of metadata to fetch.

The data returned by this endpoint can be seen here:

```
{
   "48140466-cff6-3222-bd55-63c27e43190d": {
      "artist": {
         "artist_credit_id": 65,
         "name": "Portishead",
         "artists": [
            {
               "area": "United Kingdom",
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "begin_year": 1991,
               "join_phrase": "",
               "name": "Portishead",
               "rels": {
                  "free streaming": "https://www.deezer.com/artist/1069",
                  "lyrics": "https://muzikum.eu/en/122-6105/portishead/lyrics.html",
                  "official homepage": "http://www.portishead.co.uk/",
                  "purchase for download": "https://www.junodownload.com/artists/
↪Portishead/releases/",
                  "social network": "https://www.facebook.com/portishead",
                  "streaming": "https://tidal.com/artist/27441",
                  "wikidata": "https://www.wikidata.org/wiki/Q191352",
                  "youtube": "https://www.youtube.com/user/portishead1002"
               },
               "type": "Group"
            }
         ]
      },
      "release": {
         "caa_id": 829521842,
         "caa_release_mbid": "76df3287-6cda-33eb-8e9a-044b5e15ffdd",
         "date": "1994-08-22",
         "name": "Dummy",
         "rels": [],
         "type": "Album"
      },
```

(continues on next page)

(continued from previous page)

```
      "release_group": {
         "caa_id": 829521842,
         "caa_release_mbid": "76df3287-6cda-33eb-8e9a-044b5e15ffdd",
         "date": "1994-08-22",
         "name": "Dummy",
         "rels": [],
         "type": "Album"
      },
      "tag": {
         "artist": [
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 7,
               "tag": "trip-hop"
            },
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 13,
               "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
               "tag": "trip hop"
            },
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 6,
               "genre_mbid": "cc38aba3-48ed-439a-83b9-f81a34a66598",
               "tag": "downtempo"
            },
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 5,
               "genre_mbid": "65c97e89-b42b-45c2-a70e-0eca1b8f0ff7",
               "tag": "experimental rock"
            }
         ],
         "release_group": [
            {
               "count": 6,
               "tag": "trip-hop"
            },
            {
               "count": 4,
               "genre_mbid": "cc38aba3-48ed-439a-83b9-f81a34a66598",
               "tag": "downtempo"
            },
            {
               "count": 18,
               "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
               "tag": "trip hop"
            }
         ]
      }
   }
```

(continues on next page)

(continued from previous page)

```
}
```

Parameters

- **release_group_mbids** (str) – A comma separated list of release_group_mbids
- **inc** (str) – A space separated list of "artist", "tag" and/or "release" to indicate which portions of metadata you're interested in fetching. We encourage users to only fetch the data they plan to consume.

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid release_group_mbid arguments

`GET /1/metadata/lookup/`

This endpoint looks up mbid metadata for the given artist, recording and optionally a release name. The total number of characters in the artist name, recording name and release name query arguments should be less than or equal to `MAX_MAPPING_QUERY_LENGTH`.

The data returned by this endpoint can be seen here:

```
{
   "artist_credit_name": "Rick Astley",
   "artist_mbids": [
      "db92a151-1ac2-438b-bc43-b82e149ddd50"
   ],
   "recording_mbid": "8f3471b5-7e6a-48da-86a9-c1c07a0f47ae",
   "recording_name": "Never Gonna Give You Up",
   "release_mbid": "18550a84-4ede-451c-a91a-dd9b3af9f71d",
   "release_name": "Red Hot"
}
```

Request Headers

- Authorization – Token <user token>

Parameters

- **artist_name** (str) – artist name of the listen
- **recording_name** (str) – track name of the listen
- **release_name** (str) – release name of the listen
- **metadata** (bool) – should extra metadata be also returned if a match is found, see /metadata/recording for details.
- **inc** (str) – same as /metadata/recording endpoint

Status Codes

- 200 OK – lookup succeeded, does not indicate whether a match was found or not
- 400 Bad Request – invalid arguments

Note: Because of possible abuse by AI scrapers, this endpoint now requires an auth token.

`POST /1/metadata/lookup/`

This endpoint is the POST version for looking up recording mbids and associated MusicBrainz data. It allows up to max number of items allowed. (`MAX_LOOKUPS_PER_POST` items)

A JSON document with a list of dicts each of which contain metadata (artist_name, recording_name and optionally a release name) for the recording to be looked up. The total number of characters in the artist name, recording name and release name for each recording should be less than or equal to `MAX_MAPPING_QUERY_LENGTH`.

```
{
  "recordings": [
    {
      "recording_name": "Never Gonna Give You Up",
      "artist_name": "Rick Astley",
      "release_name": "Red Hot"
    },
    {
      "recording_name": "Blinding Lights",
      "artist_name": "The Weeknd"
    }
  ]
}
```

To see what data this endpoint returns, please look at the data above for the GET version. Note that this endpoint does not support metadata and incs parameters.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – lookup succeeded, does not indicate whether a match was found or not
- 400 Bad Request – invalid arguments

Note: Because of possible abuse by AI scrapers, this endpoint now requires an auth token.

`POST /1/metadata/submit_manual_mapping/`

Submit a manual mapping of a recording messybrainz ID to a musicbrainz recording id.

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

```
{
    "recording_msid": "d23f4719-9212-49f0-ad08-ddbfbfc50d6f",
    "recording_mbid": "8f3471b5-7e6a-48da-86a9-c1c07a0f47ae"
}
```

Request Headers

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

Status Codes

- 200 OK – Mapping added, or already exists.
- 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/metadata/get_manual_mapping/`

Get the manual mapping of a recording messybrainz ID that a user added.

Request Headers

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

Status Codes

- 200 OK – The response of the mapping.
- 404 Not Found – No such mapping for this user/recording msid

Response Headers

- Content-Type – *application/json*

`GET /1/metadata/artist/`

This endpoint takes in a list of artist_mbids and returns an array of dicts that contain recording metadata suitable for showing in a context that requires as much detail about a recording and the artist. Using the inc parameter, you can control which portions of metadata to fetch.

The data returned by this endpoint can be seen here:

```
[
   {
      "area": "United Kingdom",
      "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
      "begin_year": 1991,
      "mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
      "name": "Portishead",
      "type": "Group",
      "rels": {
         "free streaming": "https://www.deezer.com/artist/1069",
         "lyrics": "https://muzikum.eu/en/122-6105/portishead/lyrics.html",
         "official homepage": "http://www.portishead.co.uk/",
         "purchase for download": "https://www.junodownload.com/artists/Portishead/
↪releases/",
         "social network": "https://www.facebook.com/portishead",
         "streaming": "https://tidal.com/artist/27441",
         "wikidata": "https://www.wikidata.org/wiki/Q191352",
         "youtube": "https://www.youtube.com/user/portishead1002"
      },
      "tag": {
         "artist": [
            {
               "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
               "count": 6,
               "genre_mbid": "cc38aba3-48ed-439a-83b9-f81a34a66598",
               "tag": "downtempo"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
                "count": 13,
                "genre_mbid": "45eb1d9c-588c-4dc8-9394-a14b7c8f02bc",
                "tag": "trip hop"
             },
             {
                "artist_mbid": "8f6bd1e4-fbe1-4f50-aa9b-94c450ec0f11",
                "count": 7,
                "tag": "trip-hop"
             }
          ]
       }
    }
]
```

Parameters

- **artist_mbids** (str) – A comma separated list of recording_mbids
- **inc** (str) – A space separated list of "artist", "tag" and/or "release" to indicate which portions of metadata you're interested in fetching. We encourage users to only fetch the data they plan to consume.

Status Codes

- 200 OK – you have data!
- 400 Bad Request – invalid recording_mbid arguments

Social

User Timeline API

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

`POST /1/user/(mb_username: 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 (either one of recording_msid or recording_mbid is sufficient):

```
{
    "metadata": {
        "recording_msid": "<The MessyBrainz ID of the recording, optional>",
        "recording_mbid": "<The MusicBrainz ID of the recording>"
    }
}
```

Parameters

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

Request Headers

- Authorization – Token <user token>

- Content-Type – *application/json*

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
- 403 Forbidden – Forbidden, you are not an approved user.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username: 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>",
    }
}
```

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/(mb_username: user_name)/timeline-event/create/review`

Creates a CritiqueBrainz review event for the user. This also creates a corresponding review in CritiqueBrainz. Users need to have linked their ListenBrainz account with CritiqueBrainz first to use this endpoint successfully.

The request should contain the following data:

```
{
    "metadata": {
        "entity_name": "<entity name, required>",
        "entity_id": "<entity id, required>",
        "entity_type": "<entity type, required>",
        "text": "<the message to post, required>",
        "language": "<language code, required>",
```

(continues on next page)

(continued from previous page)

```
        "rating": "<rating, int>",
    },
}
```

Parameters

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

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 have not linked with a CritiqueBrainz account.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/user/(mb_username: *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** (int) – If you specify a `max_ts` timestamp, events with timestamps less than the value will be returned.
- **min_ts** (int) – 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`

Request Headers

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

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.
- 403 Forbidden – Forbidden, you do not have permission to view this user's feed.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/user/(string: *user_name*)/feed/events/ int: *event_id*

Get a single feed event using the event's ID.

Parameters

- **user_name** (str) – The MusicBrainz ID of the user whose timeline is being requested.
- **event_id** (int) – ID of the event you want to fetch

Request Headers

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

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.
- 403 Forbidden – Forbidden, you do not have permission to view this user’s feed.
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`GET /1/user/(mb_username: `*user_name*`)/feed/events/listens/following`

Get feed’s listen events for followed users.

Parameters

- **user_name** (str) – The MusicBrainz ID of the user whose timeline is being requested.
- **max_ts** (int) – If you specify a `max_ts` timestamp, events with timestamps less than the value will be returned.
- **min_ts** (int) – 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`

Request Headers

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

Status Codes

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

Response Headers

- Content-Type – *application/json*

`GET /1/user/(mb_username: `*user_name*`)/feed/events/listens/similar`

Get feed’s listen events for similar users.

Parameters

- **user_name** (str) – The MusicBrainz ID of the user whose timeline is being requested.
- **max_ts** (int) – If you specify a `max_ts` timestamp, events with timestamps less than the value will be returned.
- **min_ts** (int) – 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`

Request Headers

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

Status Codes

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

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username: 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

Request Headers

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

Status Codes

- 200 OK – Successful deletion
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 401 Unauthorized – Unauthorized

- 403 Forbidden – Forbidden, you do not have permission to delete from this user's feed.
- 404 Not Found – User not found
- 500 Internal Server Error – API Internal Server Error

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username: ` *user_name*`)/feed/events/hide`

Hide events from the user feed, only recording_recommendation and recording_pin events that have been generated by the people one is following can be deleted via this endpoint. For example:

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

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

Parameters

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

Request Headers

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

Status Codes

- 200 OK – Event hidden successfully
- 400 Bad Request – Bad request, check `response['error']` for more details.
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden, you don't have permissions to hide events from this user's timeline.
- 404 Not Found – User not found
- 500 Internal Server Error – API Internal Server Error

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username: ` *user_name*`)/feed/events/unhide`

Delete hidden events from the user feed, aka unhide events. For example:

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

Request Headers

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

Status Codes

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

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username:` *user_name*`)/timeline-event/create/recommend-personal`

Make the user recommend a recording to their followers. The request should post the following data about the recording being recommended (either one of recording_msid or recording_mbid is sufficient), and also the list of followers getting recommended:

```
{
    "metadata": {
        "recording_msid": "<The MessyBrainz ID of the recording, optional>",
        "recording_mbid": "<The MusicBrainz ID of the recording>",
        "users": [], // "<usernames of the persons you want to recommend to,␣
↪required>"
        "blurb_content": "<String containing personalized recommendation>"
    }
}
```

Request Headers

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

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 personal recordings on the behalf of this user
- 403 Forbidden – Forbidden, you do not have permissions to recommend
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`POST /1/user/(mb_username: user_name)/timeline-event/create/thanks`

Make the user thank a pin/suggestion.

Request Headers

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

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 personal recordings on the behalf of this user
- 403 Forbidden – Forbidden, you do not have permissions to recommend
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

Follow API

These apis allow to interact with follow user feature of ListenBrainz.

`GET /1/user/(mb_username: user_name)/followers`

Fetch the list of followers of the user `user_name`. Returns a JSON with an array of usernames 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/(mb_username: user_name)/following`

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

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

Status Codes

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

`POST /1/user/(mb_username: user_name)/follow`

Follow the user `user_name`. A user token (found on https://listenbrainz.org/settings/) 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*

`POST /1/user/(mb_username: user_name)/unfollow`

Unfollow the user `user_name`. A user token (found on https://listenbrainz.org/settings/) 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*

Recommendations

ListenBrainz uses collaborative filtering to generate recording recommendations, which may be further processed to generate playlists for users.

Recording Recommendation API

These api endpoints allow to fetch the raw collaborative filtered recording IDs.

`GET /1/cf/recommendation/user/(mb_username: 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
            }
        ],
        "user_name": "unclejohn69",
        "count": 10,
        "total_mbid_count": 30,
        "offset": 10
    }
}
```

Note:

- This endpoint is experimental and probably will change in the future.

Parameters

- **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

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/settings/) 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/settings/) 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/(mb_username: *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*

`GET` **`/1/recommendation/feedback/user/(mb_username:`** *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"
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    "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*

Art

ListenBrainz has a (cover) art infrastructure that creates new cover art from a user's statistics or a user's instructions on how to composite a cover art grid.

As these endpoints return SVGs rather than images, you must embed them in an html `<object data="covert_art_url" type="image/svg+xml">` element rather than an `<img src="covert_art_url">` element. Otherwise external resources such as cover art images and fonts will not be loaded and the result will be useless.

See https://developer.mozilla.org/en-US/docs/Web/HTML/Element/object for reference.

POST /1/art/grid/

Create a cover art grid SVG file from the POSTed JSON data to this endpoint. The JSON data should look like the following:

```
{
        "background": "transparent",
        "image_size": 750,
        "dimension": 4,
        "skip-missing": false,
        "show-caa": false,
        "caption": false,
        "tiles": [
                "0,1,4,5",
                "10,11,14,15",
                "2",
                "3",
                "6",
                "7",
                "8",
                "9",
                "12",
                "13"
        ],
```

(continues on next page)

(continued from previous page)

```
        "release_mbids": [
                "d101e395-0c04-4237-a3d2-167b1d88056c",
                "4211382c-39e8-4a72-a32d-e4046fd96356",
                "6d895dfa-8688-4867-9730-2b98050dae04",
                "773e54bb-3f43-4813-826c-ca762bfa8318",
                "ec782dbe-9204-4ec3-bf50-576c7cf3dfb3",
                "10dffffc-c2aa-4ddd-81fd-42b5e125f240",
                "be5f714d-02eb-4c89-9a06-5e544f132604",
                "3eee4ed1-b48e-4894-8a05-f535f16a4985"
        ]
}
```

Parameters

- **background** (str) – The background for the cover art: Must be "transparent", "white" or "black".
- **image_size** (int) – The size of the cover art image. See constants at the bottom of this document.
- **dimension** (int) – The dimension to use for this grid. A grid of dimension 3 has 3 images across and 3 images down, for a total of 9 images.
- **skip-missing** (bool) – If cover art is missing for a given release_mbid, skip it and move on to the next one, if true is passed. If false, the show-caa option will decide what happens.
- **show-caa** (bool) – If cover art is missing and skip-missing is false, then show-caa will determine if a blank square is shown or if the Cover Art Archive missing image is show.

  one, if true is passed. If false, the show-caa option will decide what happens.
- **caption** (boolean) – Whether to show the release name and artist overlayed on each cover art image. Default True.
- **tiles** (list) – The tiles paramater is a list of strings that determines the location where cover art images should be placed. Each string is a comma separated list of image cells. A grid of dimension 3 has 9 cells, from 0 in the upper left hand corner, 2 in the upper right hand corner, 6 in the lower left corner and 8 in the lower right corner. Specifying only a single cell will have the image cover that cell exactly. If more than one cell is specified, the image will cover the area defined by the bounding box of all the given cells. These tiles only define bounding box areas – no clipping of images that may fall outside of these tiles will be performed.
- **release_mbids** (list) – An ordered list of release_mbids. The images will be loaded and processed in the order that this list is in. The cover art for the release_mbids will be placed on the tiles defined by the tiles parameter. If release_group_mbids are supplied as well, ONLY cover arts for release_group_mbids will be processed.
- **release_group_mbids** (list) – An ordered list of release_group_mbids. The images will be loaded and processed in the order that this list is in. The cover art for the release_group_mbids will be placed on the tiles defined by the tiles parameter. If release_mbids are supplied as well, ONLY cover arts for release_mbids will be processed.
- **cover_art_size** (int) – Size in pixels of each cover art in the composited image. Can be either 250 or 500

Status Codes

- 200 OK – cover art created successfully.

- 400 Bad Request – Invalid JSON or invalid options in JSON passed. See error message for details.

Response Headers

- Content-Type – *image/svg+xml*

See the bottom of this document for constants relating to this method.

`GET /1/art/grid-stats/(mb_username: `*user_name*`)/`*time_range*`/int: `*dimension*`/int: `*layout*`/int: `*image_size*

Create a cover art grid SVG file from the stats of a given user.

Parameters

- **user_name** (`str`) – The name of the user for whom to create the cover art.
- **time_range** (`str`) – Must be a statistics time range – see below.
- **dimension** (`int`) – The dimension to use for this grid. A grid of dimension 3 has 3 images across and 3 images down, for a total of 9 images.
- **layout** (`int`) – The layout to be used for this grid. Layout 0 is always a simple grid, but other layouts may have image images be of different sizes. See https://art.listenbrainz.org for examples of the available layouts.
- **image_size** (`int`) – The size of the cover art image. See constants at the bottom of this document.
- **caption** (`boolean`) – Whether to show the release name and artist overlayed on each cover art image. Default True
- **skip-missing** (`boolean`) – Whether to skip albums that don't have cover art, or show a placeholder. Default True

Status Codes

- 200 OK – cover art created successfully.
- 400 Bad Request – Invalid JSON or invalid options in JSON passed. See error message for details.

Response Headers

- Content-Type – *image/svg+xml*

See the bottom of this document for constants relating to this method.

`GET /1/art/(`*custom_name*`)/mb_username: `*user_name*`/`*time_range*`/int: `*image_size*

Create a custom cover art SVG file from the stats of a given user.

Parameters

- **cover_name** (`str`) – The name of cover art to be generated. See https://art.listenbrainz.org for the different types that are available.
- **user_name** (`str`) – The name of the user for whom to create the cover art.
- **time_range** (`str`) – Must be a statistics time range – see below.
- **image_size** (`int`) – The size of the cover art image. See constants at the bottom of this document.

Status Codes

- 200 OK – cover art created successfully.
- 400 Bad Request – Invalid JSON or invalid options in JSON passed. See error message for details.

Response Headers

- Content-Type – *image/svg+xml*

See the bottom of this document for constants relating to this method.

`GET /1/art/year-in-music/(int: year)/mb_username: user_name`

Create the shareable svg image using YIM stats

`POST /1/art/playlist/(uuid: playlist_mbid)/int: dimension/int: layout`

Create a cover art grid SVG file from the playlist.

Parameters

- **playlist_mbid** (`str`) – The mbid of the playlist for whom to create the cover art.
- **dimension** (`int`) – The dimension to use for this grid. A grid of dimension 3 has 3 images across and 3 images down, for a total of 9 images.
- **layout** (`int`) – The layout to be used for this grid. Layout 0 is always a simple grid, but other layouts may have image images be of different sizes. See https://art.listenbrainz.org for examples of the available layouts.

Status Codes

- 200 OK – cover art created successfully.
- 400 Bad Request – Invalid JSON or invalid options in JSON passed. See error message for details.

Response Headers

- Content-Type – *image/svg+xml*

See the bottom of this document for constants relating to this method.

Constants

Constants that are relevant to using the API:

`listenbrainz.art.cover_art_generator.MIN_IMAGE_SIZE = 128`

Minimum image size

`listenbrainz.art.cover_art_generator.MAX_IMAGE_SIZE = 1024`

Maximum image size

`listenbrainz.art.cover_art_generator.MIN_DIMENSION = 1`

Minimum dimension

`listenbrainz.art.cover_art_generator.MAX_DIMENSION = 5`

Maximum dimension

`data.model.common_stat.ALLOWED_STATISTICS_RANGE = ['this_week', 'this_month', 'this_year', 'week', 'month', 'quarter', 'year', 'half_yearly', 'all_time']`

list of allowed value for range param accepted by various statistics endpoints

Settings

These endpoints allow to customize a user's preferences/settings.

`POST /1/settings/flair`

Update a given user's flair

To remove a user's flair, pass {"flair": null}.

`POST /1/settings/timezone`

Reset local timezone for user. A user token (found on https://listenbrainz.org/settings/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – timezone reset.
- 400 Bad Request – Bad request. See error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

`POST /1/settings/troi`

Update troi preferences for the user. A user token (found on https://listenbrainz.org/settings/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – timezone reset.
- 400 Bad Request – Bad request. See error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

`POST /1/settings/brainzplayer`

Update brainzplayer preferences for the user. A user token (found on https://listenbrainz.org/settings/) must be provided in the Authorization header!

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – brainzplayer preferences updated.
- 400 Bad Request – Bad request. See error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

Miscellaneous

Various ListenBrainz API endpoints that are not documented elsewhere.

Explore API

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

GET /1/user/(mb_username: *user_name*)/fresh_releases

Get fresh releases data for the given user.

```
{
    "artist_credit_name":"Santi & Tuğçe",
    "artist_mbids":[
       "4690076b-1446-43f4-8a84-cfec56dc3601"
    ],
    "caa_id":37026686379,
    "caa_release_mbid":"9432fb06-bd84-4f2e-9386-b2f14e0de54d",
    "confidence":2,
    "release_date":"2023-10-20",
    "release_group_mbid":"30ac1ded-c254-46ac-a41f-7453dada6ae7",
    "release_group_primary_type":"Album",
    "release_group_secondary_type":null,
    "release_mbid":"9432fb06-bd84-4f2e-9386-b2f14e0de54d",
    "release_name":"The Marvelous Real",
    "release_tags":null
}
```

Parameters

- **sort** – The sort order of the results. Must be one of "release_date", "artist_credit_name", "release_name", or "confidence". Default "release_date".
- **past** – Whether to show releases in the past. Default True.
- **future** – Whether to show releases in the future. Default True.

Status Codes

- 200 OK – fetch succeeded
- 400 Bad Request – invalid date or number of days passed.

Response Headers

- Content-Type – *application/json*

GET /1/explore/fresh-releases/

This endpoint fetches upcoming and recently released (fresh) releases and returns a list of:

```
{
    "artist_credit_name": "Röyksopp",
    "artist_mbids": [
```

(continues on next page)

(continued from previous page)

```
      "1c70a3fc-fa3c-4be1-8b55-c3192db8a884"
    ],
    "release_date": "2022-04-29",
    "release_group_mbid": "4f1c579a-8a9c-4f96-92ae-befcdf3e0d32",
    "release_group_primary_type": "Album",
    "release_mbid": "1f1db316-8361-4a40-9633-550b259642f5",
    "release_name": "Profound Mysteries"
}
```

Parameters

- **release_date** – Fresh releases will be shown around this pivot date. Must be in YYYY-MM-DD format
- **days** – The number of days of fresh releases to show. Max 90 days.
- **sort** – The sort order of the results. Must be one of "release_date", "artist_credit_name" or "release_name". Default "release_date".
- **past** – Whether to show releases in the past. Default True.
- **future** – Whether to show releases in the future. Default True.

Status Codes

- 200 OK – fetch succeeded
- 400 Bad Request – invalid date or number of days passed.

Response Headers

- Content-Type – *application/json*

`GET /1/explore/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*

GET /1/explore/lb-radio

Generate a playlist with LB Radio.

Parameters

- `prompt` – The LB Radio prompt from which to generate playlists.
- `mode` – The mode that LB radio should use. Must be easy, medium or hard.

```
{
    "payload": {
        "jspf" : {}, // <JSPF playlist here>
        "feedback": [] // <user feedback items>
    }
}
```

Status Codes

- 200 OK – success
- 400 Bad Request – bad request: some parameters are missing or invalid
- 500 Internal Server Error – Troi encountered an error

Response Headers

- Content-Type – *application/json*

Donors API

GET /1/donors/recent

Get a list of most recent MeB donors with flairs.

GET /1/donors/biggest

Get a list of the biggest MeB donors with flairs.

GET /1/donors/all-flairs

Get flairs for all eligible users.

Returns a JSON object with username as key and the flair description as value. Example:

```
{
    "rob": {
        "color": "orange"
    },
    "lucifer": {
        "emoji": "devil"
    }
}
```

Status API

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.

```
{
    "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*

GET /1/status/service-status

Fetch the recently updated metrics for age of stats, dumps and the number of items in the incoming queue. This function returns JSON:

```
{
    "dump_age": 60309,
    "incoming_listen_count": 0,
    "stats_age": 38715,
    "time": 1734021912
}
```

Status Codes

- 200 OK – You have data.

Response Headers

- Content-Type – *application/json*

GET /1/status/playlist-status

Fetch the recently updated metrics for age of recommendation playlists. This function returns JSON:

```
{
    "playlists": [
        {
            "age": 55671,
            "name": "daily-jams"
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "age": 919392,
            "name": "weekly-jams"
        },
        {
            "age": 919184,
            "name": "weekly-exploration"
        }
    ],
    "time": 1734013745
}
```

Status Codes

- 200 OK – You have data.

Response Headers

- Content-Type – *application/json*

1.1.3 OpenAPI specification

Contributor rain0r went through the trouble of making an OpenAPI 3 specification for the ListenBrainz API. Many thanks! Check it out here: https://github.com/rain0r/listenbrainz-openapi

1.1.4 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[1]

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.

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

1.2 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/settings/` 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}`.

```python
from time import time
import requests

# Set DEBUG to True to test local dev server.
# API keys for local dev server and the real server are different.
DEBUG = True
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'

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(
```

(continues on next page)

(continued from previous page)

```
        url="{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": {
                "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

# Set DEBUG to True to test local dev server.
# API keys for local dev server and the real server are different.
DEBUG = True
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'
```

(continues on next page)

(continued from previous page)

```
# 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.
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="{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"],
```

(continues on next page)

(continued from previous page)

```
                                                    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
```

Lookup MBIDs

To interact with various ListenBrainz features, you will often need a MBID of the recording of a listen. You can use the *Metadata* endpoints to lookup MBID and additional metadata for the listen using its track name and artist name. For instance,

```
#!/usr/bin/env python3

import json
import requests

# Set DEBUG to True to test local dev server.
DEBUG = False
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'

def lookup_metadata(track_name: str, artist_name: str, incs: str) -> dict:
    """Looks up the metadata for a listen using track name and artist name."""
    params = {
        "recording_name": track_name,
        "artist_name": artist_name
    }
    if incs:
        params["metadata"] = True
        params["incs"] = incs
    response = requests.get(
        url="{0}/1/metadata/lookup/".format(ROOT),
        params=params
    )
    response.raise_for_status()
    return response.json()


if __name__ == "__main__":
    track_name = input('Please input the track name of the listen: ').strip()
    artist_name = input('Please input the artist name of the listen: ').strip()
    incs = input('Please input extra metadata to include (leave empty if not desired):
↪').strip()

    metadata = lookup_metadata(track_name, artist_name, incs)

    print()
    if metadata:
        print("Metadata found.")
```

(continues on next page)

(continued from previous page)

```
        print(json.dumps(metadata, indent=4))
    else:
        print("No metadata found.")
```

Please provide the prompted data to the script to lookup the given track. Currently the release argument for a listen is not used, but we plan to support in the near future, so we encourage you to start sending release information if you have it.

```
{
    "artist_credit_name": "Ariana Grande",
    "artist_mbids": [
        "f4fdbb4c-e4b7-47a0-b83b-d91bbfcfa387"
    ],
    "metadata": {
        "recording": {
            "rels": [
                {
                    "artist_mbid": "eb811bf7-4c99-4781-84c0-10ba6b8e33b3",
                    "artist_name": "Carl Falk",
                    "instrument": "guitar",
                    "type": "instrument"
                },
                {
                    "artist_mbid": "c8af4490-e48a-4f91-aef9-2b1e39369576",
                    "artist_name": "Savan Kotecha",
                    "instrument": "background vocals",
                    "type": "vocal"
                },
                {
                    "artist_mbid": "0d33cc88-28ae-44d5-be7e-7a653e518720",
                    "artist_name": "Jeanette Olsson",
                    "instrument": "background vocals",
                    "type": "vocal"
                }
            ]
        }
    },
    "recording_mbid": "9f24c0f7-a644-4074-8fbd-a1dba03de129",
    "recording_name": "One Last Time",
    "release_mbid": "be5d97b1-408a-4e95-b924-0a61955048de",
    "release_name": "My Everything"
}
```

Love/hate feedback

To provide love/hate feedback on listens, you need a recording mbid. If you do not have a recording mbid, you can look it up using the metadata endpoints. See *Lookup MBIDs* for an example of the same. Here is an example of how to submit love/hate feedback using the ListenBrainz API. Refer to *Feedback API* for more details.

```python
#!/usr/bin/env python3

import requests

# Set DEBUG to True to test local dev server.
# API keys for local dev server and the real server are different.
DEBUG = True
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'

def submit_feedback(token: str, recording_mbid: str, score: int):
    """ Submit feedback for recording. """
    response = requests.post(
        url="{0}/1/feedback/recording-feedback".format(ROOT),
        json={"recording_mbid": recording_mbid, "score": score},
        headers={"Authorization": f"Token {token}"}
    )
    response.raise_for_status()
    print("Feedback submitted.")


if __name__ == "__main__":
    recording_mbid = input('Please input the recording mbid of the listen: ').strip()
    score = int(input('Please input the feedback score (1, 0 or -1): ').strip())
    token = input('Please enter your auth token: ').strip()

    submit_feedback(token, recording_mbid, score)
```

Please provide the prompted data to the script to submit feedback.

Latest Import

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

Setting

```python
from time import time
import requests

# Set DEBUG to True to test local dev server.
# API keys for local dev server and the real server are different.
DEBUG = True
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'


def set_latest_import(timestamp, token, service="librefm"):
```

(continues on next page)

(continued from previous page)

```
    """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.

    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="{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

# Set DEBUG to True to test local dev server.
# API keys for local dev server and the real server are different.
DEBUG = True
ROOT = 'http://localhost:8100' if DEBUG else 'https://api.listenbrainz.org'

# The token can be any valid token.
TOKEN = 'YOUR_TOKEN_HERE'
```

(continues on next page)

(continued from previous page)

```python
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.

    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="{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: *Core*), `POST` a JSON document to the `submit-listens` endpoint. Submit one of three types of 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

Note: Playing Now listens are only stored temporarily. A playing now listen must be submitted again as a `single` or `import` for permanent storage.

- `import`: Submit previously saved listens
  - `payload` should contain information about *at least one* track
  - Submitting multiple listens in one request is permitted. There are some limitations on the size of a submission. A request must be less than *`MAX_LISTEN_PAYLOAD_SIZE`* bytes, and you can only submit up to *`MAX_LISTENS_PER_REQUEST`* listens per request. Each listen 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",
```

(continues on next page)

(continued from previous page)

```
      "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!"],
          "duration_ms": 222000
        },
        "artist_name": "Rick Astley",
        "track_name": "Never Gonna Give You Up",
        "release_name": "Whenever you need somebody"
      }
    }
  ]
}
```

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 ---"
    ]
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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. This should be set to playback start time of the submitted track. 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	data type	description
`release_name`	string	The name of the release this recording was played from.

The following optional elements may also be included in the `additional_info` element.

Note: If you do not have the data for any of the following fields, omit the key entirely:

Table 1: Additional Info Fields

element	data type	description
`artist_mbids`	array of strings	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`	string	A MusicBrainz Release Group ID of the release group this recording was played from.
`release_mbid`	string	A MusicBrainz Release ID of the release this recording was played from.
`recording_mbid`	string	A MusicBrainz Recording ID of the recording that was played.
`track_mbid`	string	A MusicBrainz Track ID associated with the recording that was played.
`work_mbids`	array of strings	A list of MusicBrainz Work IDs that may be associated with this recording.
`tracknumber`	string	The tracknumber of the recording. This first recording on a release is tracknumber 1.
`isrc`	string	The ISRC code associated with the recording.
`spotify_id`	string	The Spotify track URL associated with this recording. e.g.: http://open.spotify.com/track/1rrgWMXGCGHru5bIRxGFV0
`tags`	array of string	A list of user-defined folksonomy tags to be associated with this recording. For example, you can 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`	string	The name of the program being used to listen to music. Don't include a version number here.
`media_player_version`	string	The version of the program being used to listen to music.
`submission_client`	string	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`	string	The version of the submission client.
`original_submission_client`	string	If a listen was originally submitted by a different client provide the name of the client that first submitted the listen. This is useful for importers. Don't include a version number here.
`music_service`	string	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`	string	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`	string	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.
`duration_ms` and `duration`	integer	The duration of the track in milliseconds and seconds respectively. You should only include one of `duration_ms` or `duration`.
`duration_played`	integer	The duration in seconds that the user actually listened to the track.

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",
      "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"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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
↪",
              },
              "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

1.4.4 .NET

- MetaBrainz.ListenBrainz

1.4.5 Python

- pylistenbrainz

1.4.6 Java

- listenbrainz-client

1.4.7 Swift

- ListenBrainzKit

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&..`, replace it with `http://<HOSTURL>/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 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.zst`
2. `listenbrainz-listens-dump.tar.zst`
3. `listenbrainz-listens-dump-spark.tar.zst`

listenbrainz-public-dump.tar.zst

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.zst

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

listenbrainz-listens-dump-spark.tar.zst

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

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 daily.

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 all data as the next full dump.

Warning: Deleted listens present a tricky problem in this setup, since they are not included in the incremental dumps. To get a fully accurate list of listens, with deleted listens removed, you'll need to re-import a full dump.

1.7 ListenBrainz Data Update Intervals

Expected schedule:

System	Update schedule
Receiving listens, updating listen counts	Immediate[1]
Deleting listens	Removed at the top of the next hour (UTC)
Updating statistics for new listens	Daily[2]
Removing deleted listens from stats	On the 2nd and 16th of each month
Full dumps	1st and 15th of each month
Incremental dumps	Daily
Artist and track similarity	Every Sunday
Link listens	Monday morning at 2AM (UTC)
Weekly playlists	Monday morning, based on the user's timezone setting
Daily playlists[3]	Every morning, based on the user's timezone setting

Situations will occasionally arise where these take longer. If you have been a very patient user, and something still hasn't updated, check our forum for news or discussion. If you suspect something has gone very wrong you can also search for tickets, and lodge new tickets, in our ticket tracker.

This complicated schedule is caused by ListenBrainz having a lot of interconnected parts that work at different scales. For more details, read on!

[1] Listens via a connected Spotify account may not be submitted immediately, causing a short delay
[2] Statistics may take longer on the 1st and 15th of each month
[3] To generate daily playlists, follow troi-bot.

1.7.1 Listens and Listen Counts

Listen submitted via the ListenBrainz API should appear in your Dashboard immediately. The Spotify API does not immediately report completed listens; it may take several minutes for these to be reported to ListenBrainz.

Listen Count should always be up to date for listens that we have received. The only time we deviate from this is when listens are to be deleted. Deleting a listen is (surprisingly) a fairly expensive operation and we have the resources for keeping listen counts updated, but not to keep listen counts fully up to date when listens are being deleted. Any listens marked for deletion will be deleted at the top of the next hour. Once this happens, your listen counts will be consistent again.

We've spent months working on making this system work well and be scalable. This system is surprisingly complex and it took us several approaches to get to where we are now. It isn't ideal but, given our limited resources, we opted for what we call an "eventually consistent" system that focuses on being consistent (accurate) for most of our users most of the time.

1.7.2 User Statistics

Calculating user statistics is an entirely different and challenging task! For this we utilize the Spark database system (more on this below), which requires us to dump, copy and import many gigabytes of data from our Postgres database into Spark.

Currently, we use our data dumps for this purpose – we dump the full data twice a month on the 1st and 15th of the month and dump incremental dumps on a daily basis. The daily dumps update the listen data in our Spark system with new listens, but they do not remove any listens from Spark that may have been deleted from the main database since the last incremental dump. Deleted listens are removed from Spark only when we import a new full data dump on the 2nd and 16th day of the month.

For example: If you delete a listen on the 5th day of the month, you can expect that the statistics generated on the 17th will reflect the current stats of your listens as of the end of the 14th day of the month.

We recognize that this is less than ideal – we're considering how to improve this and to make the ingestion of listens and the deletion of listens both happen in real time.

1.7.3 MBID Mapper & MusicBrainz Metadata Cache

The MBID mapper is reponsible for taking an incoming listen and using nothing but artist name and recording name and finding the best match in MusicBrainz. This process normally works quite well, except for when music you'd like to listen to doesn't yet exist in MusicBrainz.

The mapper attempts to map a recording when:

1. A new listen comes in (we've never seen this listen before). If a listen is not matched, we set a timer for when to try to match the listen again. We start the timer at 1day, but for each time we fail to match it we will double the delay before try it again, up to a max of 30 days.
2. When a previously unmatched listen comes in again, we'll attempt a remapping.
3. Our periodic mapping process will go over all unmapped listens and attempt to map them. This process can take quite some time to complete (weeks at times!) and once complete it will start over again the next day.

If a listen cannot be mapped, the user can optionally tell us how to map the listen with the "Link with MusicBrainz" feature from the listen card. A few notes about this:

1. If you have an unmatched listen in your stats and then you manually link the listen, the stats will not update until the next time listens are loaded again (2nd and 16th of the month, see above)

2. If you would like to manually map a listen, but the recording does not exist in MusicBrainz, you'll need to add it there (or wait for someone else to add it). Once it has been added to MusicBrainz, it will be available for manual mappping in about 4 hours.

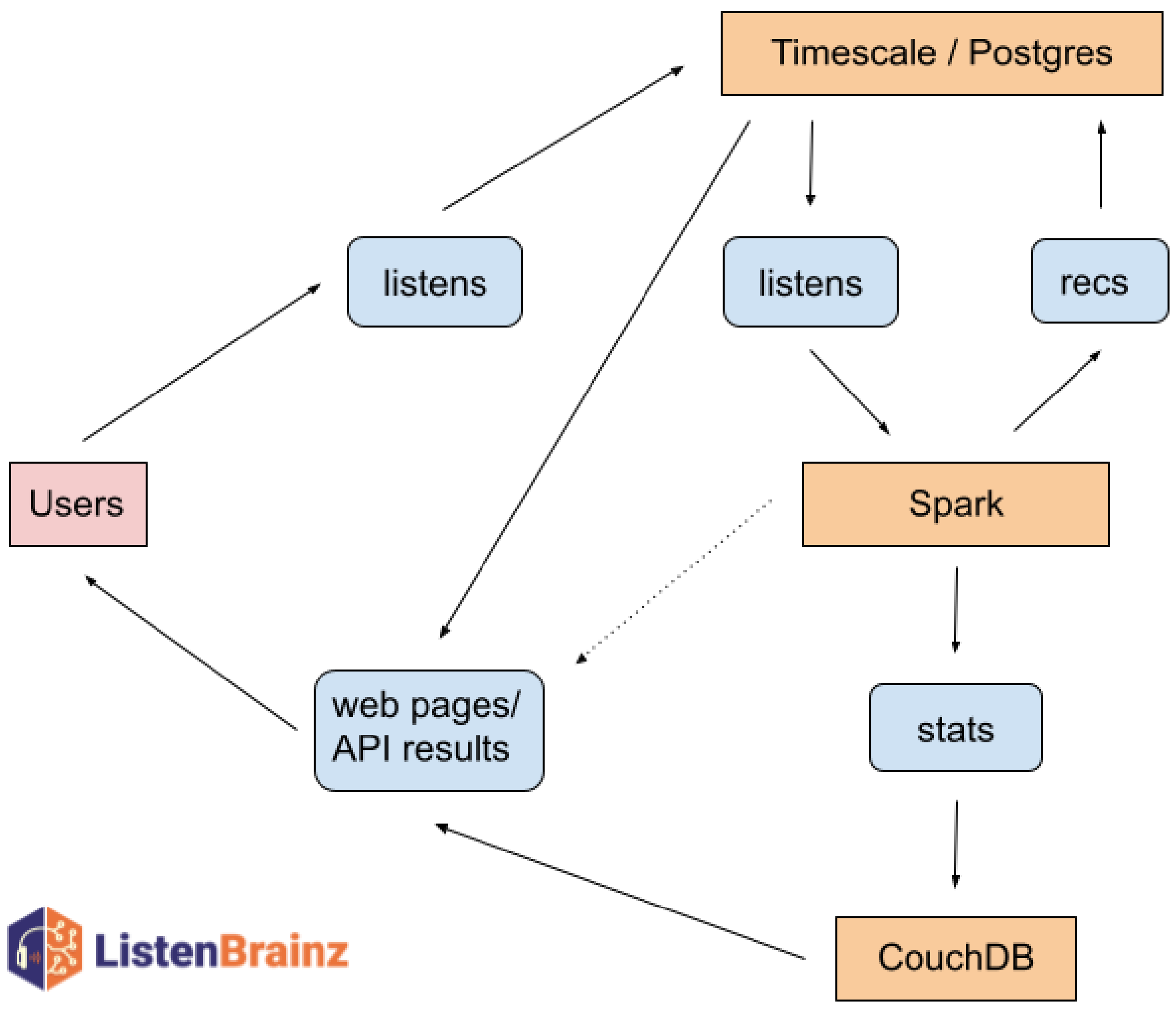
1.7.4 ListenBrainz data infrastructure

The ListenBrainz project makes use of three major types of databases: Postgres/Timescale, Spark and CouchDB. Postgres is our primary workhorse that serves user data (accounts, followers, likes/hates, etc). Timescale (an extended version of Postgres) is used to store Listens and Playlists. Postgres and Timescale are fantastic tools for providing a specific piece of information quickly (e.g which users does this a user follow). However Postgres/Timescale are not great for inserting large amounts of data into the database each day – it slows everything down for everyone. So we store the computed user listening statistics in CouchDB, which is suitable for ingesting large volumes of data and serving it for a day, before it is replaced with the next iteration of the data.

Finally, we have Spark, which is a batch processing system. Spark is designed to work with large datasets in a batch fashion, where the data for *all* users might be processed in one batch task. Spark works with huge amounts of data in one go, which is very distinct from our use of Postgres/Timescale and CouchDB.

All of the tools we use are excellent open source tools. Each does a fantastic job, at the tasks they were designed for. There is no one open source solution for all of our needs, so we shuffle data from one system to another as we need it. This, however, brings latency and delays in keeping all of our data up to date.

Take a look at the general overview of how data flows between each of these systems:



In the future we hope to serve content (stats) directly from our Spark installation (with the help of existing tools) as indicated by the dotted arrow between Spark and the API/web pages box. This will further reduce the latency of some of our services.

1.8 Server development

1.8.1 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

1.8.2 Clone listenbrainz-server

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

Note: Windows users are advised to clone the repository in their WSL2 file system to avoid code watcher issues. Please refer to *run docker inside WSL* for more information.

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

1.8.3 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.

1.8.4 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:8100/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
```

Now, open the new config.py file (don't change config.py.sample) with your favorite text editor.

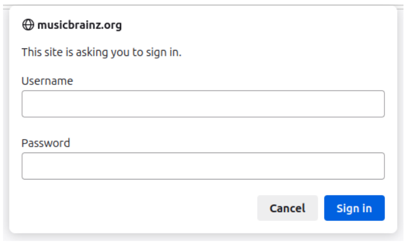
Note: If you are accessing your development server using a port other than `8100`, ensure that you update the `SERVER_ROOT_URL` to reflect the appropriate port number. If you are accessing your development server using a host other than `localhost` (e.g., GitHub Codespaces), ensure that you uncomment and update `SERVER_NAME`, and ensure `SERVER_ROOT_URL` is updated accordingly to maintain consistency and support the appropriate host details.

Next look for this section in the file.

```
# MusicBrainz OAuth
OAUTH_CLIENT_ID = "CLIENT_ID"
OAUTH_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.

Note: Make sure the `OAUTH_CLIENT_ID` and `OAUTH_CLIENT_SECRET` parameters are set properly, failing to do so will result in a basic browser auth popup like the one below:



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:8100`.

To use the Spotify importer you need to register an application on the Spotify Developer Dashboard. Use `http://127.0.0.1:8100/settings/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:8100/` as the homepage URL and `http://localhost:8100/settings/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 = f'{SERVER_ROOT_URL}/settings/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.

To use Funkwhale for music playback and scrobbling, you need to set up the callback URL in the configuration file. Funkwhale uses OAuth for authentication with each Funkwhale instance independently.

```
# FUNKWHALE
FUNKWHALE_CALLBACK_URL = f'{SERVER_ROOT_URL}/settings/music-services/funkwhale/callback/'
```

To use Navidrome for music playback, you need to set up an encryption key for securely storing Navidrome passwords. Generate a base64 encoded encryption key and add it to the configuration.

To generate an encryption key, run the following Python command:

```
python3 -c "from cryptography.fernet import Fernet; print(Fernet.generate_key().decode())
↪"
```

```
# NAVIDROME
NAVIDROME_ENCRYPTION_KEY = ""
```

Update the `NAVIDROME_ENCRYPTION_KEY` field with the generated key.

Note: The encryption key is used to securely encrypt and decrypt Navidrome passwords stored in the database. Make sure you do not change this key, or you will have to disconnect and reconnect to your Navidrome account.

1.8.5 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.

1.8.6 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_ts_db --create-db
```

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

1.8.7 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 8100. 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:
- "8100:80"
```

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:8100
```

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.

1.8.8 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
```

1.8.9 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. Note that all tests are run: Unit tests and integration tests.

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 [path-to-tests-file-or-directory]  # run specific 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
```

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.

1.8.10 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
```

1.8.11 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
```

1.9 Spark development

The ListenBrainz Spark environment is used for computing statistics and computing recommendations. 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.

1.9.1 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
```

1.9.2 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!

1.9.3 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*.

1.9.4 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.

1.9.5 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*.

1.9.6 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.

1.10 Architecture

1.10.1 Services

This is a list of the docker containers for ListenBrainz services used in local development and running in the MetaBrainz server infrastructure.

In production, webservers run uwsgi server to serve the flask application. In development, the flask development server is used.

Table 3: Webservers

Development	Production	Description
web	listenbrainz-web-prod	serves the ListenBrainz flask app for the website and APIs (except compat APIs).
api_compat	listenbrainz-api-compat-prod	serves a flask app for only Last.fm compatible APIs.
websockets	listenbrainz-websockets-prod	runs websockets server to handle realtime listen and playlist updates.

Table 4: Databases and Cache

Development	Production	Description
redis	listenbrainz-redis	redis instance used for caching all stuff ListenBrainz.
lb_db	listenbrainz-timescale	timescale instance for ListenBrainz to store listens and playlists. in development environment, the all databases are part of *lb_db* container.
lb_db	postgres-floyd	primary database instance shared by multiple MetaBrainz projects. The main ListenBrainz DB resides here as well as the MessyBrainz DB.

Table 5: Misc Services

Development	Production	Description
timescale_write	listenbrainz-timescale-writer-prod	runs timescale writer which consumes listens from incoming rabbitmq queue, performs a messybrainz lookup and inserts listens in the database.
spotify_reader	listenbrainz-spotify-reader-prod	runs a service for importing listens from spotify API and submitting to rabbitmq.
spark_reader	listenbrainz-spark-reader-prod	processes incoming results from spark cluster like inserting statistics in database etc.
rabbitmq	rabbitmq-clash	rabbitmq instance shared by MetaBrainz services. listenbrainz queues are under /listenbrainz vhost.

Table 6: Only Production Services

Production	Description
listenbrainz-labs-api-prod	serves a flask app for experimental ListenBrainz APIs
listenbrainz-api-compat-nginx-prod	runs a nginx container for the compat API that exposes this service on a local IP, not through gateways.
listenbrainz-cron-prod	runs cron jobs used to execute periodic tasks like creating dumps, invoking spark jobs to import dump, requesting statistics and so on.
exim-relay-listenbrainz.org	smtp relay used by LB to send emails.
listenbrainz-typesense	typesense (typo robust search) used by the mbid-mapping.
listenbrainz-mbid-mapping	A cron container that fires off periodic MBID data processing tasks.
listenbrainz-mbid-mapping-writer-prod	Maps incoming listens to the MBID mapping as well as updating the mapping.
listenbrainz spark cluster	spark cluster to generate statistics and recommendations for LB.

1.10.2 Listen Flow

Listens can be submitted to ListenBrainz using native ListenBrainz API, Last.fm compatible API (API compat) and AudioScrobbler 1.2 compatible API (API compat deprecated). Each api endpoint validates the listens submitted through it and sends the listens to a RabbitMQ queue based on listen type. Playing Now listens are sent to the Playing Now queue, and permanent listens are sent to the Incoming queue.

Playing now listens are ephemeral are only stored in Redis, with an expiry time of the duration of the track (if duration is unavailable then a configurable fallback time is used). The Playing now queue is consumed by Websockets service. The frontend connects with the Websockets service to display listens on the website without manually reloading the page.

On the other hand, "Permanent" Listens need to be persisted in the database. Timescale Writer service consumes from the Incoming queue. It begins with querying the MessyBrainz database for MessyBrainz IDs. MessyBrainz tries to find an existing match for the hash of the listen in the database. If one exists, it is returned otherwise it inserts the hash and data into the database and returns a new MessyBrainz ID.

Once the writer receives MSIDs from MessyBrainz, the MSID is added to the track metadata and the listen is inserted in the listen table. The insert deduplicates listens based on a (user, timestamp, track_name) triplet i.e. at a given timestamp, a user can have a track entry only once. As you can see, listens of different tracks at the same timestamp are allowed for a user. The database returns the "unique" listens to the writer which publishes those to Unique queue.

The Websockets server consumes from the unique queue and sends a list of tracks to connected clients (like the now playing queue). The MBID mapper also consumes from the unique queue and builds a MSID->MBID mapping using these listens.

1.10.3 Frontend Rendering

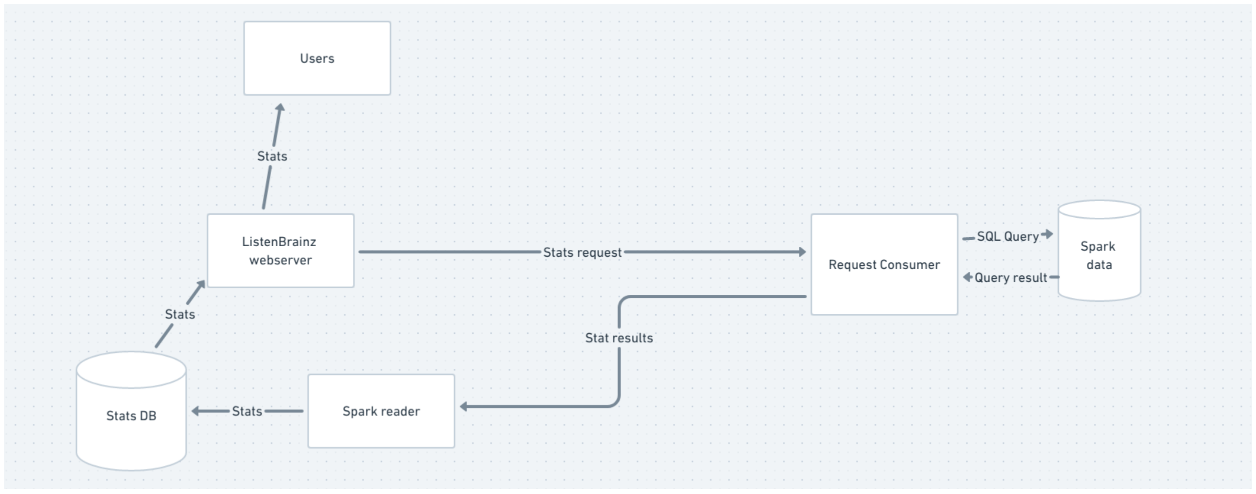
ListenBrainz frontend pages are a blend of Jinja2 templates (Python) and React components (Javascript). The Jinja2 templates used are bare bones , they include a placeholder div called *react-container* into which the react components are rendered. To render the components, some data like current user info, api url etc are needed. These are injected as json into two script tags in the HTML page, to be consumed by the React application: page-react-props and global-react-props.

Most ListenBrainz pages will have a Jinja2 template and at least 1 React component file. The components are written in Typescript, and we use Webpack to transpile them to javascript, to compile CSS from LESS and to minify and bundle everything. In local development, this is all done in a separate Docker container *static_builder* which watches for changes in front-end files and recompiles automatically. In production, the compilation happens only once and at time of building the docker image.

Using script tags, we manually specify the appropriate compiled javascript file to include on a given page in its Jinja2 template.

1.11 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.

1.11.1 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
```

1.12 MBID Mapping

The MBID mapping scripts allow us to take metadata from the messybrainz database and look up recording MBIDs from the MusicBrainz database.

Note: The MBID Mapping source code lives in `listenbrainz/mbid_mapping` but is run independently from the main listenbrainz web docker image. You can use your own virtual environment or use `listenbrainz/mbid_mapping/build.sh` to build a standalone docker image.

1.12.1 Database tables

The MBID Mapping supplemental tables hold preprocessed data from the MusicBrainz database.

- `mapping.canonical_musicbrainz_data`: The MBID and Name of Recordings, Artists (and credits), and Releases for all recordings in MusicBrainz
- `mapping.canonical_recording_redirect`: A mapping to find the "canonical" recording given an artist credit + recording name
- `mapping.canonical_release_redirect`: A mapping to find the "canonical" release given an artist credit + release name

These tables can be populated by running

```
python mapper/manage.py canonical-data
```

The update process build the new data in a temporary table and then replaces them in a single transaction. This means that lookups can continue to run on the existing tables while the new ones are being built.

1.12.2 Fuzzy lookups

We use typesense as a way of performing quick, fuzzy lookups based on artist name and recording name

Build the typesese index with

```
python mapper/manage.py build-index
```

As with the data tables, a new typesense collection is created and then swapped into place in a single operation.

Build the mapping tables and then the typesense index directly afterwards with

```
python mapper/manage.py create-all
```

1.12.3 MBID Mapper

The mapper looks for new MSIDs submitted to messybrainz and finds a matching MBID in MusicBrainz

```
    python3 -u -m listenbrainz.mbid_mapping_writer.mbid_mapping_writer
```

A background thread pushes items to be processed onto a queue - recent submissions first, and then if nothing is to be done, old items. The processing thread pops items off the queue and then looks them up, adding them to the `mbid_mapping` table.

There is also a background thread that fires off daily, which looks for listens that have been written to the listens table, but for some reason do not have a matching mapping entry. (This could happen due to restarts or problems with the mapper itself). These are called legacy listens.

The background thread will walk the entire listens table once a day to find these legacy listens and attempt to map them. In the same thread we also look for mapping items with timestamp of the unix epoch (1970-01-01 00:00:00), which indicates that they ought to be re-checked. Currently we have no automated mechanism in place for setting any mapping entries to the epoch.

TODO: Detuning algorithm TODO: match quality types

1.13 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 ...`.

1.13.1 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]...
```

add_missing_to_listen_users_metadata

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

clear-expired-do-not-recommends

Delete expired do not recommend entries from database

```
./develop.sh manage clear-expired-do-not-recommends [OPTIONS]
```

delete-old-user-data-exports

Delete old and expired user data exports

```
./develop.sh manage delete-old-user-data-exports [OPTIONS]
```

delete_listens

Complete all pending listen deletes and also run update script for updating listen metadata since last cron run

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

delete_pending_listens

Complete all pending listen deletes since last cron run

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

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_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.

notify_yim_users

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

Options

`--year` `<year>`

Year for which to send the emails

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-top-manual-mappings

Refresh top manual msid-mbid mappings view

```
./develop.sh manage refresh-top-manual-mappings [OPTIONS]
```

run-daily-jams

Generate daily playlists for users soon after the new day begins in their timezone. This is an internal LB method and not a core function of troi.

```
./develop.sh manage run-daily-jams [OPTIONS]
```

Options

`--create-all`

Create the daily jams for all users. if false (default), only for users according to timezone.

run-metadata-cache-seeder

Query external services' new releases api for new releases and submit those to our cache as seeds

```
./develop.sh manage run-metadata-cache-seeder [OPTIONS]
```

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-msid-tables

Scan tables using msids to find matching mbids from mapping tables and update them.

```
./develop.sh manage update-msid-tables [OPTIONS]
```

update_db

Updates the datbase by running the specified SQL file at FILENAME

```
./develop.sh manage update_db [OPTIONS] FILENAME
```

Arguments

FILENAME
  Required argument

update_user_emails

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

update_user_listen_data

Scans listen table and update listen metadata for all users

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

1.13.2 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.

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

Options

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

-lp, --location-private `<location_private>`
path to the directory where the private dumps 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`**
: If True, make a listens dump

`--spark`, **`--no-spark`**
: If True, make a spark listens dump

`--db`, **`--no-db`**
: If True, make a public/private postgres dump

`--timescale`, **`--no-timescale`**
: If True, make a public/private timescale dump

`--stats`, **`--no-stats`**
: If True, make a couchdb stats dump

`-lt`, **`--location-temp`** `<location_temp>`
: path to directory to use for creating necessary temporary files during dumps.

`-lpt`, **`--location-private-temp`** `<location_private_temp>`
: path to directory to use for creating necessary temporary files during private dumps.

create_incremental

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

Options

`-l`, **`--location`** `<location>`

`-t`, **`--threads`** `<threads>`

`--dump-id` `<dump_id>`

create_mbcanonical

Create a dump of the canonical mapping tables. This includes the following items:

- metadata for canonical recordings
- canonical recording redirect
- canonical release redirect

These tables are created by the mapping *canonical-data* management command.

- If canonical-data is called with –use-lb-conn then the canonical metadata and recording redirect tables will be in the listenbrainz timescale database connection
- If called with –use-mb-conn then all tables will be in the musicbrainz database connection.
- The canonical release redirect table will always be in the musicbrainz database connection.

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

Options

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

--use-lb-conn, --use-mb-conn
: Dump the metadata table from the listenbrainz database

create_sample

Create a sample data dump.

```
./develop.sh manage dump create_sample [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

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 sample_archive (str): the path to the ListenBrainz sample 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>
  the path to the ListenBrainz listen dump archive to be imported

-s, --sample-archive <sample_archive>
  the path to the ListenBrainz sample dump archive to be imported

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

1.13.3 ListenBrainz Spark

These commands are used to interact with the Spark Cluster.

./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_popularity

```
./develop.sh manage spark cron_request_popularity [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]
```

cron_request_similarity_datasets

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

request_compact_listens

Send a request to spark cluster to compact listens imported from listenbrainz

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

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_entity_stats

Send an entity stats request to the spark cluster

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

Options

--type <type_>

Required Type of statistics to calculate

Options
listeners

--range <range_>

Required Time range of statistics to calculate

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

--entity <entity>

Entity for which statistics should be calculated

Options
artists | release_groups

--database <database>

Name of the couchdb database to store data in

request_fresh_releases

Send the cluster a request to generate release radar data.

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

Options

--days <days>

Number of days of listens to consider for artist listening data

--database <database>

Name of the couchdb database to store data in

--threshold <threshold>

Number of days of listens to consider for artist listening data

request_import_deleted_listens

Send a request to spark cluster to import deleted listens from listenbrainz

```
./develop.sh manage spark request_import_deleted_listens [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

--use-local

Use local dump instead of FTP

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

--use-local

Use local dump instead of FTP

request_import_mlhd_dump

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

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

request_import_pg_tables

Send the cluster a request to import metadata table from MB db postgres

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

request_import_sample

Send the cluster a request to import a sample dump

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

request_missing_mb_data

Send the cluster a request to generate missing MB data.

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

Options

--days <days>
    Request missing musicbrainz data based on listen data of given number of days

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.

--use-transformed-listencounts
    Whether to apply a transformation function on the listencounts or use original listen playcounts

request_per_artist_popularity

Request mlhd popularity data using the specified dataset.

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

Options

--use-mlhd

Use MLHD+ data or ListenBrainz listens data

--entity <entity>

Required

Options

recording | release | release_group

request_popularity

Request mlhd popularity data using the specified dataset.

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

Options

--use-mlhd

Use MLHD+ data or ListenBrainz listens data

--entity <entity>

Required

Options

artist | recording | release | release_group

request_recommendations

Send the cluster a request to generate recommendations.

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

Options

--raw <raw>

Generate given number of raw recommendations

--user-name <users>

Generate recommendations for given users. Generate recommendations for all users by default.

request_recording_discovery

Send the cluster a request to generate recording discovery data.

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

request_similar_artists

Send the cluster a request to generate similar artists index.

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

Options

--days <days>

Required The number of days of listens to use.

--session <session>

Required The maximum duration in seconds between two listens in a listening session.

--contribution <contribution>

Required The maximum contribution a user's listens can make to the similarity score of a artist pair.

--threshold <threshold>

Required The minimum similarity score to include a recording pair in the simlarity index.

--limit <limit>

Required The maximum number of similar artists to generate per artist (the limit is instructive. upto 2x artists may be returned than the limit).

--skip <skip>

Required the minimum difference threshold to mark track as skipped

--production

Required whether the dataset is being created as a production dataset. affects how the resulting dataset is stored in LB.

request_similar_recordings

Send the cluster a request to generate similar recordings index.

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

Options

--days <days>
: The number of days of listens to use. required if using listens data

--use-mlhd
: Use MLHD+ data or ListenBrainz listens data

--session <session>
: **Required** The maximum duration in seconds between two listens in a listening session.

--max-contribution <max_contribution>
: **Required** The maximum contribution a user's listens can make to the similarity score of a recording pair.

--threshold <threshold>
: **Required** The minimum similarity score to include a recording pair in the simlarity index.

--limit <limit>
: **Required** The maximum number of similar recordings to generate per recording (the limit is instructive. upto 2x recordings may be returned than the limit).

--skip-threshold <skip_threshold>
: **Required** the minimum difference threshold to mark track as skipped

--only-stage2
: whether to reuse existing outputs of intermediate chunks

--production
: **Required** whether the dataset is being created as a production dataset. affects how the resulting dataset is stored in LB.

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_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 | era_activity | artist_evolution_activity

`--range <range_>`

    Required Time range of statistics to calculate

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

`--entity <entity>`

    Entity for which statistics should be calculated

    Options
    artists | releases | recordings | release_groups

request_tags

Generate the tags dataset with percent rank

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

request_troi_playlists

Bulk generate troi playlists for all users

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

Options

`--slug <slug>`

    Required

    Options
    weekly-jams | weekly-exploration

`--create-all`

    whether to create the periodic playlists for all users or only for users according to timezone.

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 | artist_evolution_activity | era_activity | genre_activity

--range <range_>

Required Time range of statistics to calculate

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

--entity <entity>

Entity for which statistics should be calculated

Options
artists | releases | recordings | release_groups

--database <database>

Name of the couchdb database to store data in

request_year_in_music

Send the cluster a request to generate all year in music statistics.

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

Options

--year <year>

Year for which to calculate the stat

--import-pg-tables, --no-import-pg-tables

whether to import the pg tables before generating the stats.

request_yim_artist_evolution

Calculate artist evolution activity for Year in Music

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

Options

--year <year>
Year for which to calculate artist evolution

request_yim_day_of_week

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

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

Options

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

request_yim_genre_activity

Calculate genre activity for Year in Music

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

Options

--year <year>
Year for which to calculate genre activity

request_yim_listen_count

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

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

Options

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

request_yim_listening_time

Send request to calculate yearly total listening time stat for each user to the spark cluster

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

Options

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

request_yim_listens_per_day

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

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

Options

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

request_yim_most_listened_year

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

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

Options

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

request_yim_new_artists_discovered

Send request to calculate count of new artists user listened to this year.

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

Options

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

request_yim_new_release_stats

Send request to calculate new release stats to the spark cluster

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

Options

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

request_yim_similar_users

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

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

Options

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

request_yim_top_discoveries

Send the cluster a request to generate tracks of the year data and then once the data has been imported generate YIM playlists.

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

Options

--year <year>
Year for which to generate the playlists

request_yim_top_genres

Send request to calculate top genres each user listened to this year.

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

Options

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

request_yim_top_missed_recordings

Send the cluster a request to generate tracks of the year data and then once the data has been imported generate YIM playlists.

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

Options

--year <year>
Year for which to generate the playlists

request_yim_top_stats

Send request to calculate top stats to the spark cluster

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

Options

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

1.14 Troubleshooting

1.14.1 Docker Installations

Windows

If changes to JS files are not being watched or hot reloaded by the host file system, follow these steps:

1. Clone or move the project into your WSL2 file system.
2. Create a `.wslconfig` file under `C:/Users/<user-name>/` with the following content:

```
[wsl2]
localhostforwarding=true
```

3. To apply the changes, you may need to shut down the WSL 2 VM by running `wsl --shutdown` in the command prompt. Then, restart your WSL instance.

For more detailed information, refer to the wsl settings page.

1.15 Production Deployment

Note: This documentation is for ListenBrainz maintainers for when they deploy the website

1.15.1 Cron

You can cleanly shut down cron from `docker-server-configs` by running

```
./scripts/terminate_lb_cron.sh
```

If no cron jobs are running, this will stop and delete the cron container. If a job is running it will notify you and not stop the container.

1.16 Building Docker Images

Note: This documentation is for ListenBrainz maintainers for when they deploy the website

1.16.1 Production Images

When a Github release is made, production images are automatically built and pushed by the Publish image action. The git tag associated with the Github release is used as docker image tag.

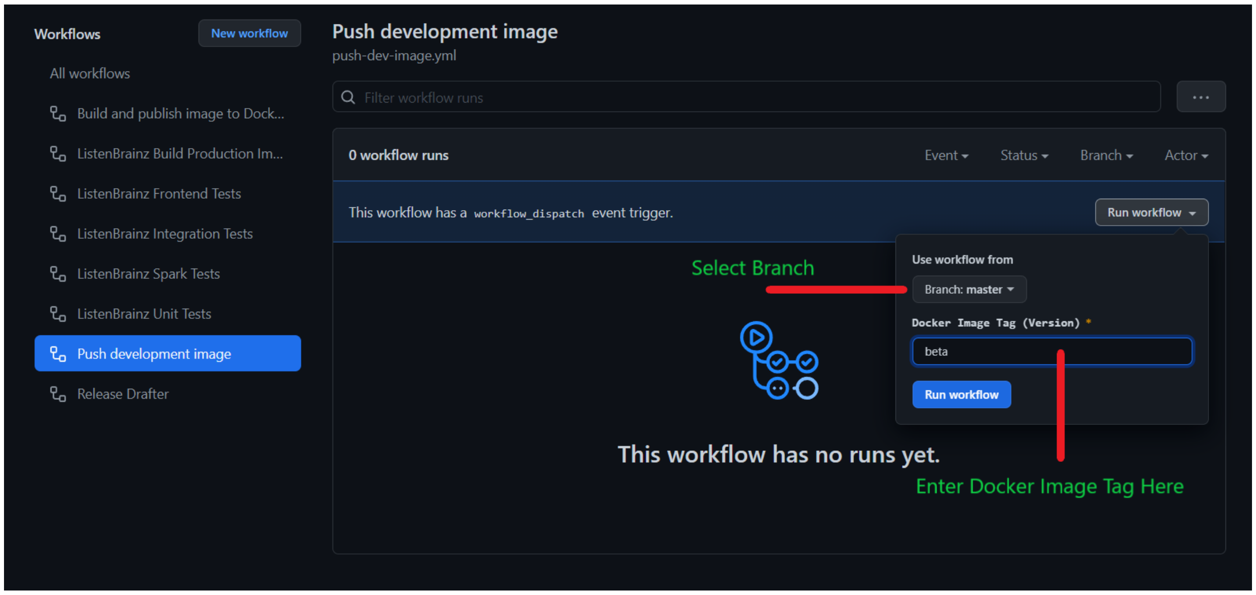
1.16.2 Test Images

From time to time we want to build images to test PRs on beta.listenbrainz.org or test.listenbrainz.org. To build images for this purpose you can either use the *docker/push.sh* script or Github Actions.

Note: Usually, the tags for these images is test or beta. However, you can use any arbitrary image tag. This is useful if you want to test multiple PRs simultaneously or avoid conflicting with another developer's images. These image tags appear on Dockerhub forever unless removed manually. To my knowledge it is not an issue. Regardless its not a bad idea to login to Dockerhub once in a while and clean up such unused test tags.

1.16.3 Using Github Actions

1. Go to Actions -> Push deployment image.
2. Select the branch and enter the docker image tag (version).



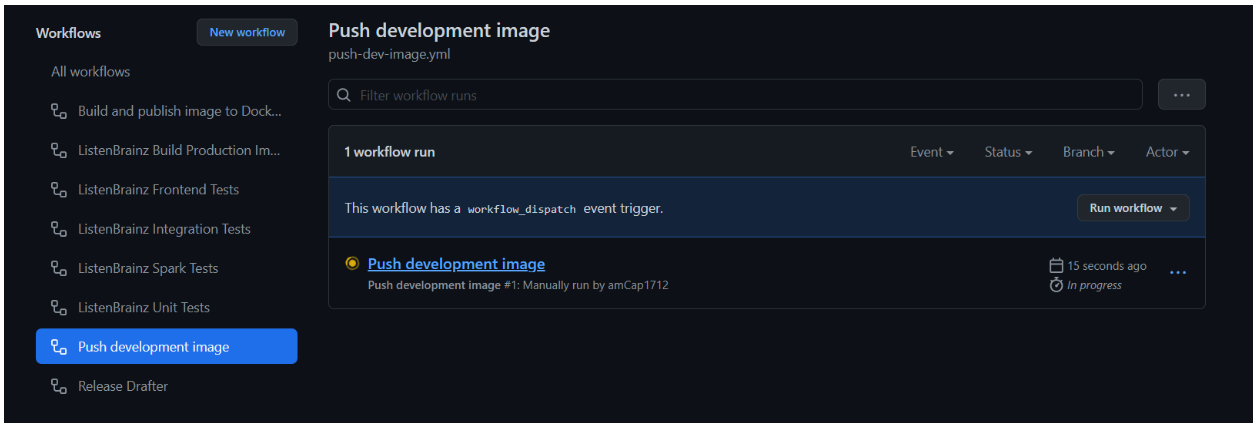
This workflow has a `workflow_dispatch` event trigger.

Branch: master

Docker Image Tag (Version) *

beta

1. Click on Run Workflow.
2. The image will be built and pushed to Docker Hub with the desired tag.
3. To monitor the status of the build, wait for the workflow run to appear. You may need to wait for a few seconds and reload the page.



This workflow has a `workflow_dispatch` event trigger.

Push development image

Push development image #1: Manually run by amCap1712

1.16.4 Using docker/push.sh script

If Github Actions is unavailable or you want to take advantage of local docker build cache, you can use the *docker/push.sh* script. You will need to be correctly authenticated to docker hub to push this image. From the repository root, invoke the script with desired docker image tag. For example:

```
./docker/push.sh beta
```

1.17 Data Dumps

1.17.1 Check FTP Dumps age script

Dumps may fail in production due to many reasons. We have a script to check the latest dump available on the FTP is younger than a specified timeframe. If the latest dump is older, an email is sent to the maintainers. This email is usually responsible for bringing dump failures to the notice of maintainers. This script is part of the ListenBrainz cron jobs and is scheduled to run a few hours after the regular dump times. If dumps are not working but no email was received by the maintainers, it is possible that the cron jobs are not setup properly.

1.17.2 Logs

Looking at the logs is a good starting point to debug dump failures, the log file is located at `/logs/dumps.log` inside the listenbrainz-cron-prod container. The output of dump-related jobs is redirected in the crontab . Open a bash shell in the cron container by running `docker exec -it listenbrainz-cron-prod bash`.

This file is large, so use **`tail`** instead of **`cat`** to view the logs. For example: `tail -n 500 /logs/dumps.log` will list the last 500 lines of the log file.

From the log file, you should probably be able to see whether the error occurred in python part of the code or bash script. If you see a python stack trace, it is likely that sentry recorded the error too. The sentry view sometimes offers more details so searching sentry for this error can be helpful.

1.17.3 Manually triggering dumps

If you want to re-run a dump after it fails, or manually trigger a dump then you can run the dump script manually. A few things need to be kept in mind while doing this, the *create_full* invoked to do the dump accepts a `--dump-id` parameter to number the dump. If no id specified, the script will look in the database for the last id, add 1 to it and use it for the dump.

```
select * from data_dump order by created desc;
```

If a dump failed too early in the script, it won't have an id in the database. Otherwise, it will have created one before failing. To be sure, check the `data_dump` table in the database. If the id exists and the dump had failed , it makes sense to reuse that dump id when generating the dump again manually.

Also the bash script to create dumps performs setup, cleanup and syncing to FTP tasks so do not invoke the python command directly. The bash script forwards arguments to the python command so you can pass any arguments that the python command accepts to it as well. See the current version of the script in the repository for more details. Here is an example of how you can manually specify the id of the dump (copied the cronjob command at the time of writing and added the argument before redirecting):

```
flock -x -n /var/lock/lb-dumps.lock /code/listenbrainz/admin/create-dumps.sh incremental␣
↪--dump-id 700 >> /logs/dumps.log 2>&1
```

Note: Full dumps take over 12 hours to complete. If you run the command directly and close the terminal before full dumps completion, the dumps will get interrupted and fail. So either run the command inside a **`tmux`** session or use a combination of **`nohup`** and **`&`** with the dump command.

1.18 MBID Mapping

For a background on how the mapping works, see *MBID Mapping*

1.18.1 Containers

The mapping tools run in two containers:

- **`mbid-mapping-writer-prod`: Populates the `mbid_mapping` table for new listens. Built from the**
  main ListenBrainz dockerfile.
- **`mbid-mapping`: Periodically generates the MBID Mapping supplemental tables, typesense index,**
  and huesound index. Built from `listenbrainz/mbid_mapping/Dockerfile`

1.18.2 Data sources

In the production environment, the `mbid-mapping` container reads from the MB replica on aretha.

1.18.3 Debugging lookups

If a listen isn't showing up as mapped on ListenBrainz, one of the following might be true:

- The item wasn't in musicbrainz at the time that the lookup was made
- There is a bug in the mapping algorithm

If the recording doesn't exist in MusicBrainz during mapping, a row will be added to the `mbid_mapping` table with the MSID and a `match_type` of `no_match`. Currently no_match values aren't looked up again automatically.

You can test the results of a lookup by using *https://labs.api.listenbrainz.org/explain-mbid-mapping <https://labs.api.listenbrainz.org/explain-mbid-mapping>* This uses the same lookup process that the mapper uses. If this returns a result, but there is no mapping present it could be due to data being recently added to MusicBrainz or improvements to the mapping algorithm.

If no data is returned or an incorrect match is being returned, this should be reported to us, by adding a comment to *LB-1036 <https://tickets.metabrainz.org/browse/LB-1036>*.

In this case you can retrigger a lookup by seting the `mbid_mapping.last_updated` field to '1970-01-01 00:00:00' (the unix epoch). The mapper will pick up these items and put them on the queue again.

```
UPDATE mbid_mapping SET last_updated = 'epoch' WHERE recording_msid = '00000737-3a59-
↪4499-b30a-31fe2464555d';
UPDATE mbid_mapping SET last_updated = 'epoch' WHERE match_type = 'no_match' AND last_
↪updated = now() - interval '1 day';
```

In the LB production environment these items will be picked up and re-processed once a day.

1.19 Debugging Spotify Reader

To debug spotify reader issues, begin with checking logs of the container. The ListenBrainz admin panel has external_service_ouath and listens_importer table which show the user's token, importer error if any, last import time and latest listen imported for that user.

Sometimes spotify's recent listens API does not show updated listens for hours while the currently playing endpoint does. So the user may see currently playing listens arrive but the "permanent" listens missing. To confirm this is the case, you can use the spotify api console and directly query the api to see what listens spotify is currently returning. You can get the user's spotify access token for this endpoint from admin panel. If the api does not have listens, it makes sense those to not be present in ListenBrainz yet. However if the api returns the listens but those are not in ListenBrainz, there is likely an issue with Spotify Reader. Consider adding more logging to the container to debug issues.

1.20 RabbitMQ

1.20.1 Maintenance

Tolerance to connectivity issues

RabbitMQ is a mandatory service required by consul-template config used in common by almost all ListenBrainz containers. Therefore, ListenBrainz will refuse to come up if no RabbitMQ instance is running. If an instance is available but there are connectivity issues, various ListenBrainz services will remain up but throw errors while trying to perform some functions.

The most important part that relies on RabbitMQ is the listens submission API. If RabbitMQ is unreachable, users will be unable to submit listens to ListenBrainz.

Maintenance mode

It doesn't exist. To perform maintenance operations, ListenBrainz requires switching to another instance of RabbitMQ to prevent any data loss, even for a short period of time.

Data importance

ListenBrainz uses RabbitMQ in various places, for a brief overview *see listen flow*. The most important is listens submission. Listens are published to the *incoming* exchange and expected to be persisted durably until the timescale writer has acknowledged writing those to the database. This data is of utmost importance.

Other uses of RabbitMQ in ListenBrainz include delivering now playing listens to websockets, unique listens to the mbid mapping writer, results from spark cluster to the database etc. In these cases, the data can be regenerated. Data loss in these cases is tolerable as long as it is known that some messages were lost.

Data persistence

Messages are expected to be processed within seconds (or minutes during activity peaks), but because of the importance of the listen data a persistent volume is needed. Listen data messages are critical and should be backed up, other messages can be regenerated and can be ignored in case of a disaster.

Procedures

- Start service: LB containers automatically connect to RabbitMQ on startup.
- Reload service configuration: Update the RabbitMQ service details in consul configuration for LB and deploy a new image.
- Restart service: Restart LB docker containers and each container will disconnect and reconnect to RabbitMQ.
- Move service:
  - Create vhost, user, permissions, queues in the new instance
  - Stop LB producers (except the web and api containers)
  - Use shovels to transfer existing messages from old RabbitMQ instance to new one
  - Build an image using the a new consul config pointing to new RabbitMQ instance
  - Deploy all consumers using the new image
  - Deploy all producers using the new image
  - Stop shovels

  There will be no data loss but a short downtime while the containers restart.
- Remove service: LB cannot function without RabbitMQ. So the only way is to stop LB containers, and LB will become unavailable.

1.20.2 Implementation details

- Connectivity issues are reported through both Docker logs and Sentry.
- ListenBrainz has multiple producers and consumers.
- message protocol version: AMQP 0.9.1.
- heartbeat timeout: client sets to 0, rabbitmq will use the server specified value.
- ack mode:
  - producers do not use any ack mode.
  - auto ack: *spark-request-consumer-michael*
  - manual ack: all other consumers
- Each connection identifies itself with RabbitMQ server by using the name of the docker container in which the service is running.

1.21 Updating Production Database Schema

Warning: The production database cluster is serious business . Think twice whenever interacting with it and check with others in face of the slightest doubt.

The listenbrainz image on which most of ListenBrainz containers run has the **`psql`** command installed. You can *exec* into a container and use the **`psql`** to connect to the relevant database and execute scripts. The connection parameters to connect to the databases are in `/code/listenbrainz/listenbrainz/config.py`.

Whenever modifying the database, run the sql commands inside a transaction if possible. Once you have started the transaction, execute the commands you want to. Do not commit the transaction yet. Double check the state of the database to ensure the changes are in line with what you expect. If so commit the transaction otherwise rollback and contact other maintainers.

1.22 Pull Requests Policy

It is recommended that maintainers (unless the change is urgently needed) do not push directly or merge pull requests without review . By default, **one** approving review is sufficient to merge a pull request. The pull request author or the reviewer can request more reviews or review from a specific person as they deem necessary.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

INDEX