About the API

Our web API is still in early development phase and its functionality might be changed at any point. The API is based on standard REST principles, right now only GET Request are implemented.

Please Note: This is not a public API

This API is solely used internally by Universal Music. This means it allows actions that would not be available in public APIs. While normally you would prevent possibilities for SQL Injections at all time, in this API you can officially execute SQL Code. While “DROP DATABASE” will of course not work (incoming SQL Code will not directly be executed), you can run calculations that could bring down the database server. This is by design, since we want to run complex queries without needing to add code to the API.

Using this vulnerability is not an achievement. We trust that you use the API responsible and with care.

Also the API is still under active development, some aspects may not yet work as expected.

Request limits

The API is limited to 60 requests per 1 min

Authentication

Right now authentication is only possible through “http authentication”. This means in the header of the request you should add:

```
Authorization: Basic b64encode("username:password")
```

(“username:password” needs to be base64 encoded)

```
https://en.wikipedia.org/wiki/Basic_access_authentication
```

API Endpoints

There are two base endpoints “Artist” and “Audience”. Artist represent all information about a music artist. Audience includes target group information about a specific artist. Artists and its information can be access with our Artist Cluster ID or a Spotify artist id. The “related_artists” endpoints returns a list of related artists of the queried artists.

/Audience/{artist cluster id}
/Audience/{artist cluster id}/{category}

/Artist
/Artist/{artist cluster id}
/Artist/{spotify artist id}

/Artist/{artist cluster id}/related_artists
/Artist/{spotify artist id}/related_artists

Filter and search options

The filter and search system uses a SQL-like language which is interpreted, restructured and send to a DB Server. Fields can be filtered just like in a "WHERE" part of a SQL Query.
Available fields:

id, spotify_id, followers, popularity, gravity, label, caption, modularity, genre_name

For example: To search for artists named Coldplay:
http://umgcluster.elasticbeanstalk.com/api/Artist?caption=Coldplay
To search for artists with more than 40,000 followers and a gravity above 0.2
http://umgcluster.elasticbeanstalk.com/api/Artist?followers>40000&gravity>0.2
but the URL needs to be encoded which makes it
http://umgcluster.elasticbeanstalk.com/api/Artist?followers%3E40000&gravity%3E0.2
(Most browsers do the encoding automatically)
%3E = > and %3C = <

There are some additional functions that are not included in standard SQL:

AC_AVG	Average
AC_COUNT	Count
AC_MAX	Maximum
AC_MIN	Minimum
AC_SUM	Sum
AC_STDDEV	Standard Derivation
AC_VARIANCE	Variance

For example: Get all artists with more followers than average.
http://umgcluster.elasticbeanstalk.com/api/Artist?followers%3EAC_AVG(followers)

Other SQL operators can be used also:

Caption is either Madonna, Eminem or Coldplay
caption IN (Madonna, Eminem, Coldplay)
http://umgcluster.elasticbeanstalk.com/api/Artist?caption%20IN%20(Madonna,%20Eminem,%20Coldplay)

Name of artists begins with Mad
caption LIKE 'Mad%'
http://umgcluster.elasticbeanstalk.com/api/Artist?caption%20LIKE%20%27Mad%25%27

Followers is bigger than the Average + one standard derivation
followers > AC_AVG(followers) + AC_STDDEV(followers)
http://umgcluster.elasticbeanstalk.com/api/Artist?followers%20%3E%20AC_AVG(followers)%20%2B%20AC_STDDEV(followers)

Artists which name sounds like Eminem using Double Metaphone
dmetaphone('Eminem')=dmetaphone(caption)
http://umgcluster.elasticbeanstalk.com/api/Artist?dmetaphone(%27Eminem%27)=dmetaphone(caption)

Additional keywords

limit	Limit the output to x records (std: 50)
offset	Used for paging, starte with record number x (std: 0)
gettotal	Possible Values {0,1} return the number of total records for this query
orderby	Order by this field
orderdir	Direction of ordering {ASC,DESC}
as	Return in a different file format, implemented options: xls
view	Determines which fields are returned and which format is used (see Views) (std: default)

Views

Views allow the user to only get the information that they really need. You can define which fields will be returned, if you want them as key/value pairs or just an array and if the output is indented. You can also use it to transform the output into the format you want (limit decimal places, round etc.).

Views are organized in types (For the type of data they structure, right now those types are Artist, Audience. Every view needs to have a unique name. When adding a view to an existing name the view gets updated.

The following endpoints can be used to manipulate views:

GET	/views	Return all views
GET	/view/{type}/{name}	Return view by type and name
DELETE	/view/{type}/{name}	Deletes view by type and name
PUT	/view	Creates/Updates view (specified in JSON)

To create/update a view use the following structure:

```
{
    "name": "newView",
    "type": "Artist",
    "view" : {
      "fields" : {
        "id" : {
          "Alias" : "id"
        },
        "spotify_id" : {
          "Alias" : "spotify_id"
        },
        "caption" : {
          "Alias" : "artist_name"
        },
        "label" : {
          "Alias" : "label"
        },
        "modularity" : {
          "Alias" : "modularity_class"
        },
        "followers" : {
          "Alias" : "spotify_followers"
        },
        "popularity" : {
          "Alias" : "spotify_popularity"
```

```
        },
        " to_char(gravity,'FM0.00')" : {
          "Alias" : "gravity"
        },
        "last_updated" : {
          "Alias" : "last_updated"
        }
      },
      "print_field_names" : true,
      "beautify" : true,
      "order" : [ ]
    }
}
```

Fields is a key/value object with the field you want to select as key and another key/value object as value containing the “Alias” object that will determine the key in the json output. The key can be used to format the values by using SQL function. In the example, the gravity field will be returned as a string with 2 digest instead of a float.

“print_field_names” determines if the output will be key/value or a plain array of values.

If “beautify” is true the output will be indented.

“order” is right now not used.