Web Technology: Spotify

Sam Verkoelen
Media Technology
Leiden University
sdverkoelen@gmail.com

Donna Piët
Media Technology
Leiden University
donnapiet@gmail.com

Jules Verdijk
Media Technology
Leiden University
jules.ver@gmail.com

ABSTRACT
Spotify is a popular online music streaming service with over 20 million songs. This paper discusses the Spotify technology and the possibilities to integrate this technology into applications. Topics such as the metadata API, Apps and the Spotify URI scheme are covered. Furthermore, the strengths and weaknesses of the Spotify technology are discussed, as well as intended and surprising applications. Finally, a 'getting started' guide is provided, with a basic explanation of using the metadata API in an application.

1. PURPOSE, CONTEXT AND HISTORY

Spotify is an online music streaming service which was created in Stockholm, Sweden, and launched in 2008. With a database of more than 20 million songs, with more than 20,000 songs added per day, Spotify is one of the biggest music streaming services around, available in 56 countries [15]. Creating a Spotify account allows users to create playlists and subsequently share these playlists with other users and collaboratively edit them. The sharing of these playlists is supported by many websites, which link to the playlist in Spotify. Users with an account can also integrate it with their Facebook and twitter accounts which allows them to share tracks and playlists or access those of their friends.

- 2006: Spotify is developed in Stockholm, Sweden, by founders Daniel Ek and Martin Lorentzon. It announces licensing deals with a number of major music labels [26].
- 2008 (October): Spotify is launched [34].
- 2010 (September): Spotify has 10 million users, of which a quarter are paying users [13].
- 2011 (August): Spotify announces an Software Development Kit (SDK) with which iOS developers can access its database to build apps [28].
- 2011 (November): Spotify launches its first number of apps [24].
- 2012 (February): Spotify throws a party, “Music Hack Weekends”, to attract app developers.
- 2013 (March): Spotify has 24 million users, of which 6 million were paying users [30].
- 2014 (May): Spotify has 40 million users, of which 10 million were paying users [27].

2. OPERATING PRINCIPLES

The Spotify service is primarily a streaming music service. As mentioned above, users can listen to over 20 million tracks on multiple devices. However, an internet connection is required to do so because Spotify is a streaming service. This means that the user does not own the music files but plays them directly from a Spotify server. In this process every music file is chopped into little pieces, these pieces are sent to the user in a continuous flow (stream) of data and played in the right order at the end user. This is a positive thing from the perspective of the Spotify user since they do not need to have a local copy of every song they might want to play. The benefit for Spotify itself is that they stay in control over the content, for example serving higher quality

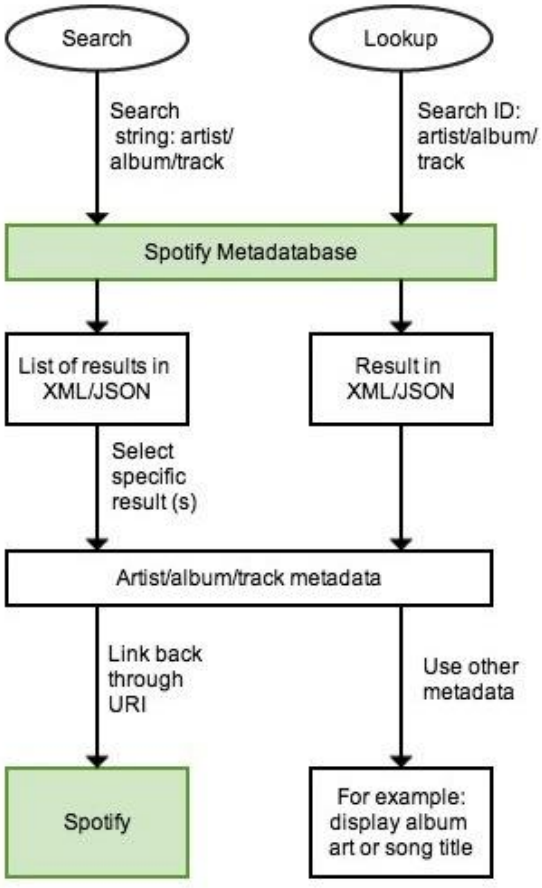

Figure 1. Workflow of the Spotify Metadata API.

files to premium users. Nevertheless, streaming does also have a downside. The internet connection and available bandwidth determine whether the service can be used. Also, for every time that a user plays a song Spotify will have to request the files from the server. Therefore more data will be transferred over the internet in comparison to a regular download.

A solution to minimize the amount of traffic from the server is *caching*. When the user plays a song and the chopped stream of files flows from the server to a device, a local copy is made and saved for 30 days [14]. This cached copy will then be accessed the next time the user plays that same song, instead it being downloaded again. Notwithstanding, for premium users it is possible to keep the cache for 3.333 tracks per device [18].

Inside the Spotify application there is also the possibility to run *Apps*. Apps are small applications mostly created by third party developers that run inside the Spotify application. For app developers there is the possibility to make use of the Spotify database and user specific information [25]. An example app that uses this information is the 'Songkick App' [23]. Songkick looks at recently listened artists and notifies the user when this artist gives a concert nearby. Nonetheless, Spotify has announced that it will soon discard its apps and will come with an alternative in the future [4].

For developers it is possible to incorporate a *direct Uniform Resource Identifier (URI) link* to the Spotify application. This is possible because Spotify is a registered protocol within an URI scheme [3, 33]. This enables direct links from, for example, a website to the Spotify application such as:

```
<a href="spotify:track:id">track</a>
```

When a user clicks this link the browser will know that it has to open Spotify and start playing that specific track.

Furthermore, developers are also able to request data from Spotify through their *Metadata Application Programming Interface (API)* [19]. This API is a web service that makes it possible to request specific metadata about artists, albums or tracks in a structured way. The API currently has two main functions called *search* and *lookup*, where the latter is the function that actually serves the developer the metadata. This metadata includes information like the duration of a track or the date on which an album was released. The search function provides the developer with a way of translating a human-readable string (i.e. the name of a band) into a Spotify specific identifier (ID) which then can be used to lookup the metadata. The data, by default, is served in the widely supported Extensible Markup Language (XML) format. However, it is also possible to receive the data in JavaScript Object Notation (JSON) formatting.

3. STRENGTHS AND WEAKNESSES

For the regular user there are several alternatives to Spotify that offer comparable music streaming services. These services differ mostly in their availability in certain regions, the size of the music library, alternative music browsing (like apps, channels and podcasts), cost and social media integration [21]. Some of the best known alternatives that offer services similar to Spotify are: Deezer, Rdio, BeatsMusic and Google Music.

For developers focusing on integrating Spotify in a new (web-)environment or application, alternatives can differ greatly from Spotify. Listed are some of the strengths and weaknesses of using Spotify and its API, which are then compared to some of the aforementioned alternatives.

3.1. Strengths

- A low volatility music database, which enables the caching of requested metadata. Spotify offers the option to see whether cached data has changed, so data will not have to be requested again [19].
- A linking system using Spotify-specific URIs, enabling direct linking to in-app tracks, albums,

Service	Format	Search	Data	Authorization	Method
Spotify [19]	XML, JSON	Artist, Album, Track	Limited	None	GET
Rdio [10]	XML, JSON	Album, Artist, Label, Playlist, Track, User	Full	Developer Account	GET, POST, DELETE
Deezer [6]	XML, JSON, JSONP	Artist, Album, Track	Limited	None	GET, POST, DELETE
BeatsMusic [1]	JSON	Artist, Track, Album, Playlist, User, Genre	Only name and ID	User ID	GET, POST, DELETE

Table 1. Functionalities and limitations of the metadata API's of Spotify and alternative music streaming services.

artists, playlists and users [17].

- Customizable widgets to embed tracks or playlists in HyperText Markup Language (HTML) web pages, which will play in a Spotify client [31].
- The metadata API is accessible without authorization.
- The metadata API provides additional information, like track length, availability, explicit content and popularity.

3.2. Weaknesses

- All core functionalities (such as music playback and making playlists) can only be used within a Spotify application and cannot be implemented in other applications.
- The metadata API only provides data; storing or saving data is not possible.
- The metadata API is restricted to searching for albums, artists and tracks [22].
- The metadata API has a request rate limit of ten requests per Internet Protocol (IP) Address per second [19].

3.3. Alternatives

In terms of music streaming services that resemble Spotify

there are some differences in the services offered to possible developers. For an overview of their metadata API's see Table 1.

In requesting metadata, Spotify is relatively restricted in regard to its alternatives. Rdio and BeatsMusic offer more search options and Rdio opens up almost all its data to developers. Noticeable are some other possibilities these services offer. All three offer the possibility to edit and store data through an API, ranging from basic playlist editing and ratings (all three) to accessing and editing almost all personal user data (Rdio, BeatsMusic).

Looking past metadata and into using the core functionalities of streaming, Deezer, Rdio and BeatsMusic offer embedded out-of-app music streaming to logged in users.

- Deezer offers a JavaScript SDK, which incorporates the metadata and offers custom User Interface (UI) design [7].
- BeatsMusic offers a JavaScript SDK [9].
- Rdio offers a flash player with a ActionScript 3 or a JavaScript interface [8].
- Spotify does offer an embedded javascript web player, but this player activates streaming within the Spotify app, the embedded player can be used to pause, play and skip. This player uses Spotify URIs to control the streaming [31].

Integrating Spotify in new environments and applications is limited to accessing basic metadata and linking to in-app functionalities. Alternatives also give access to metadata, but are not restricted to in-app functionalities, and offer the possibility to actually integrate music streaming within new environments. Nonetheless, all services require a user to be registered and logged in with their own account. When choosing which service to use in developing, it is recommended to take into account its population of registered users in your region.

4. TYPICAL APPLICATIONS

4.1. Searching in Spotify

Developers have created alternative ways of browsing Spotify data using their metadata API. One example of this is the site 'Spotify Music Catalog' , where you can check whether a certain artist is available on Spotify [29]. These applications generally search for a song, artist or album in the metadata and either return a formatted table of the results or simply state whether the searched object is available on Spotify. Another example is the Spotify Search add-on for the Mozilla Firefox browser [32]. Once installed, this 'Spotify-search' add-on allows a user to highlight any text (for example the name of a band or song in an article), right-click, and press 'Search in Spotify'. The add-on takes the highlighted text and converts it to a Spotify URI to link to Spotify and automatically search.

4.2. Linking to Spotify

Another common implementation of Spotify is the direct linking to in-app artists, albums, tracks or playlists. For example, the site 'NME' uses a Spotify-widget to link directly to in-app playlists that are related to recent articles [20]. Festivals also often post a Spotify URI link to a playlist with their attending bands on their website as promotion. These URI linkings can also be found in social media posts, for example, on twitter streams of artists and labels.

5. SURPRISING APPLICATIONS

Spotify only offers developers a limited amount of access to their data. Therefore the possibilities of what developers can do with this data in their own applications is also quite limited and most often results in apps using their own databases and subsequently linking back to Spotify for playback. Despite the limited amount of options, a number of developers have still found creative ways of using the metadata API and Spotify URI.

5.1. Creating Playlists

'Spartify' allows multiple users on different devices to create a playlist together, for example, at a party. Creating playlists together is a service that is already included within Spotify, nonetheless this requires every user to be logged in to Spotify. Spartify creates a playlist on its own server and lets users add songs to the communal playlist through their website [2]. Each user can add a track, and Spartify will search for the track in Spotify and play that track, keeping

track of the play length of each track, and then directly link to the next track. Thus the whole playlist functionality is done on the server of Spartify and Spotify is used solely to play each track through an URI link [2].

5.2. Adding Music to Your Spotify

Rather than adding your albums to Spotify by searching for them one by one, 'Covify' created and application that lets you add your albums by holding them in front of your webcam. Using its own database of album covers, Covify then searches and plays your album in Spotify [5].

5.3. Discovering New Music

One feature which is not available on Spotify is the ability to play a random song. An example of a site which offers this possibility is http://www.karnhuset.net [16]. Another surprising way of discovering new music is the application 'Forgotify'. Forgotify filters the Spotify data by popularity, which is based on amount of plays, and then plays a song which is the least popular [12].

6. GETTING STARTED

To begin working with the Spotify technology it is useful to be able to test Spotify URIs. To do so, download the Spotify application and log in [11]. Further requirements for this chapter are a web server (we use Apache), PHP Hypertext Preprocessor (PHP) with a minimum version of 5.3, simpleXML module for PHP and the possibility for your firewall to make outgoing connections. A basic knowledge of HTML is required for this example.

6.1. Linking: Understanding the Spotify URI

There are three types of links described in the Spotify URI specification document. The first one (lookup) is used when the Spotify ID for the artist, album or track that will be requested is known.

```
//Format
spotify:<artist|album|track>:<id>


//Example
spotify:track:6PZDPg3dZgJkNL6nVMUB4b
```

The second case (search) is when a Spotify ID is unknown, but there is a human-readable string such as artist, album or track available.

```
//Format
spotify:search:<text>

//Example
spotify:search:beatles
```

The last case is a method to open a specific user's playlist. A playlist ID is required in this case. It is noteworthy that searching for an ID is not possible. However, users can copy this URI directly from the Spotify application.

```
//Format
spotify:user:<username>:playlist:<id>

//Example
spotify:user:sdverkoelen:playlist:1fSKe4IOL0m1ONXZ
yYihzq
```

How to link to one of these URIs depends on your programming or markup language. In the case of HTML it is as simple as:

```
<a
href="spotify:user:sdverkoelen:playlist:1fSKe4IOL0
m1ONXZyYihzq">Webtech playlist</a>
```

6.2. Metadata API: Search for a Link

The following example will show a PHP page with a search form that searches in the Spotify database. The first step is to create an empty HTML page with the title "Spotify search", and save this file as *search.php*.

```
<html>
    <head>
        <title>Spotify search</title>
    </head>
    <body>


    </body>
</html>
```

The next step is to create a form and place it inside the body tags. Since this form will point to itself we place the filename search.php in the action parameter. In this example a GET is used as the method because it passes the search term in the URL, making it possible to share or bookmark the page. The entered search term is passed under the name 'q', this name has been chosen in order to be consistent with the API, which also uses 'q' as the search term.

```
   <body>
    <form action="search.php" method="get">
     <input type="text" name="q" value="" />
     <input type="submit" name="" value="search"
/>
    </form>
   </body>
```

In order to make the search process a bit more user friendly it is possible to place any previous search terms inside the textfield, if there are any.

```
<input type="text" name="q" value="<?php
if(isset($_GET['q'])) echo($_GET['q']) ?>" />
```

The actual code that accesses the API can be placed below our form inside the <?php ?> tags. However, before a request is sent to the API, there needs to be a check to make sure that GET-variable 'q' even exists and that it holds a value.

```
//Only search when there is an input
if(isset($_GET['q']) && $_GET["q"] != ''){
}
```

Inside this 'if statement', the URL of the API can be

constructed together with the GET-variable ‘q’.

```
//The API search URL

$url = 'http://ws.spotify.com/search/1/album?q='.
$_GET['q'];
```

This URL can be loaded directly into simpleXML, an optional module of PHP capable of handling XML files. The result (an object of the complete XML file) will be stored in the variable $xml.

```
//load the API XML file

$xml = simplexml_load_file($url);
```

The Spotify API will return every single search result inside <track> tags and the simpleXML will convert it into an object. Therefore a loop through all the tracks can be created using the $xml->track variable.

```
//Looping through all the tracks

foreach($xml->track as $track){

}
```

Because this example will show the links in a structured way, a table should be placed around the `<?php ?>` tags. For neatness, a heading is added in this table showing which column contains the track, album or artist data.

```
<table width="100%">
    <tr>
        <td width="40%">Track</td>
        <td width="40%">Album</td>
        <td width="20%">Artist</td>
    </tr>
<?php
    //Only search when there is an input
    if(isset($_GET['q']) && $_GET["q"] != ''){
        //The API search URL
        $url =
        'http://ws.spotify.com/search/1/album?q='.
        $_GET['q'];

        //load the API XML file
        $xml = simplexml_load_file($url);

        //Looping through all the tracks
        foreach($xml->track as $track){
        }
    }
?>
</table>
```

Inside this loop the `$track` variable represents the object of a single search result. Different variables such as the track name and a link to the album can be accessed in this way:

```
$track->name   //track name

$track->album->attributes()->href //link to album
```

With the use of basic HTML and the `$track` object it is now possible to construct links to the track, artist and album.

```
//Show and link to track

echo('<td><a href="'.$track->attributes()-
>href.'">'.$track->name.'</a></td>');


//Show and link to album

echo('<td><a href="'.$track->album->attributes()-
>href.'">'.$track->album->name.'</a></td>');


//Show and link to artist

echo('<td><a href="'.$track->artist->attributes()-
>href.'">'.$track->artist->name.'</a></td>');
```

When these links are placed inside the *foreach loop*, the finished example should look like this:

```
<html>
 <head>
  <title>Spotify search</title>
 </head>
  <body>
   <form action="search.php" method="get">
    <input type="text" name="q" value="<?php
if(isset($_GET['q'])) echo($_GET['q']) ?>" />
    <input type="submit" name="" value="search" />
   </form>
   <table width="100%">
    <tr>
     <td width="40%">Track</td>
     <td width="40%">Album</td>
     <td width="20%">Artist</td>
    </tr>
    <?php

     //Only search when there is an input
     if(isset($_GET['q']) && $_GET["q"] != ''){

     //The API search URL
     $url =
'http://ws.spotify.com/search/1/album?q='.
$_GET['q'];

     //load the API XML file
     $xml = simplexml_load_file($url);
```

```
      //Looping through all the tracks
      foreach($xml->track as $track){
       echo('<tr>');

       //Show and link to track
       echo('<td><a href="'.$track->attributes()->href.'">'.$track->name.'</a></td>');

       //Show and link to album
       echo('<td><a href="'.$track->album->attributes()->href.'">'.$track->album->name.'</a></td>');

       //Show and link to artist
       echo('    <td><a href="'.$track->artist->attributes()->href.'">'.$track->artist->name.'</a></td>');
       echo('</tr>');
                 }
              }
          ?>
    </table>
  </body>
</html>
```

The final web page can be seen in figure 2.

6.3. Metadata API: Search for a Link

In the following example a PHP page will be built with an artist ID form that searches in the Spotify database.

The first step is to create an empty HTML page with the title "Spotify lookup" and save this file as *search.php*.

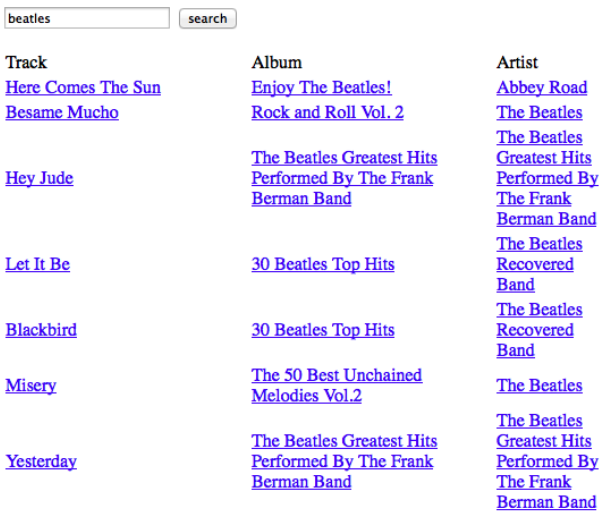

Figure 2. Screenshot of the search page

```
<html>
    <head>
        <title>Spotify lookup</title>
    </head>
    <body>


    </body>
</html>
```

The next step is to create a form and place it inside the body tags. Since this form will point to itself, we place the filename *lookup.php* in the action parameter. In this example a GET is used as the method because it passes the search term in the URL, making is possible to share or bookmark the page. The entered search term is passed under the name 'id', this name has been chosen in order to be consistent with the API, which also uses 'id' as the search term.

```
<body>
  <form action="lookup.php" method="get">
    <input type="text" name="id" value="" />
    <input type="submit" name="" value="search" />
  </form>
</body>
```

In order to make the search process a bit more user friendly, it is possible to place any previous search terms inside the textfield, if there are any.

```
<input type="text" name="id" value="<?php
if(isset($_GET['id'])) echo($_GET['id']) ?>" />
```

The actual code that accesses the API can be placed below our form inside the `<?php ?>` tags. However, before a request is sent to the API there needs to be a check to make sure that the GET-variable 'id' even exists, and that it holds a value.

```
//Only search when there is an input
if(isset($_GET['id']) && $_GET["id"] != ''){


}
```

Inside this *if statement* the URL of the API can be constructed together with the GET-variable 'id'.

```
//The API lookup URL
$url =
'http://ws.spotify.com/lookup/1/?uri=spotify:artist:'.$_GET['id'].'&extras=albumdetail';
```

This URL can be loaded directly into simpleXML, an optional module of PHP capable of handling XML files. The result (an object of the complete XML file) will be stored in the variable `$xml`.

```
//load the API XML file
$xml = simplexml_load_file($url);
```

The Spotify API will return every single search result inside `<albums><album /></albums>` tags, and the simpleXML will convert it into an object. Therefore, a loop through all the albums can be created using the `$xml->albums->album` variable.

```
//Looping through all the tracks
foreach($xml->albums->album as $album){


}
```

Because this example will show the links in a structured way, a table should be placed around the `<?php ?>` tags. For neatness, a heading is added in this table showing which column contains the album, release or availability data.

```
<table width="100%">
    <tr>
        <td width="30%">Album</td>
        <td width="5%">Released</td>
        <td width="65%">Availability</td>
    </tr>
<?php
    //Only search when there is an input
    if(isset($_GET['id']) && $_GET["id"] != ''){
        //The API search URL
        $url =
        'http://ws.spotify.com/lookup/1/?uri=spoti
        fy:artist:'.$_GET['id'].'&extras=albumdeta
        il';


        //load the API XML file
        $xml = simplexml_load_file($url);


        //Looping through all the tracks
        foreach($xml->albums->album as $album){
        }
    }
?>
</table>
```

Inside this loop the `$track` variable represents the object of a single search result. Different variables such as the track name and a link to the album can be accessed in this way.

```
$track->name   //track name
$track->album->attributes()->href //link to album
```

With the use of basic HTML and the `$album` object it is now possible to construct links to the track, artist and album.

```
//Show and link to track
echo('<td><a href="'.$track->attributes()-
>href.'">'.$track->name.'</a></td>');


//Show and link to album
echo('<td><a href="'.$track->album->attributes()-
>href.'">'.$track->album->name.'</a></td>');


//Show and link to artist
echo('<td><a href="'.$track->artist->attributes()-
>href.'">'.$track->artist->name.'</a></td>');
```

When these links are placed inside the *foreach loop*, the finished example should look like this.

```
<html>
 <head>
   <title>Spotify lookup</title>
 </head>
   <body>
    <form action="lookup.php" method="get">
     <input type="text" name="id" value="<?php
    if(isset($_GET['id'])) echo($_GET['id']) ?>"
    />
      <input type="submit" name="" value="get the
data" />
     </form>
     <table width="100%">
      <tr>
       <td width="30%">Album</td>
       <td width="5%">Released</td>
       <td width="65%">Availability</td>
      </tr>
      <?php
        //Only search when there is an input
        if(isset($_GET['id']) && $_GET["id"] !=
        ''){
           //The API lookup URL
           $url =
           'http://ws.spotify.com/lookup/1/?uri=spo
           tify:artist:'.$_GET['id'].'&extras=album
           detail';


           //load the API XML file
           $xml = simplexml_load_file($url);


           //Looping through all the tracks
           foreach($xml->albums->album as $album){
            echo('<tr>');
       //Show and link to album
       echo('    <td><a href="'.$album-
```

```
        >attributes()->href.'">'.$album-
        >name.'</a></td>');


        //Show release year

         echo('<td>'.$album->released.'</td>');


        //Show availability

        echo('    <td>'.$album->availability-
        >territories.'</td>');


        echo('</tr>');

                  }

              }

         ?>

      </table>

  </body>

</html>
```

The final web page can be seen in figure 3.

7. FINAL THOUGHTS

With more than twenty million tracks, and new ones being added daily, Spotify offers users a comprehensive music streaming service through a range of apps on desktop, mobile and the web. By means of a directly accessible metadata API, Spotify offers developers several methods to gather and use metadata about all the tracks, artists and albums in their music library. Actual music streaming is limited to the apps, and requires users to run such an app. To access this functionality as a developer, Spotify offers a Spotify-specific URI which can be used to activate certain in-app functionalities through direct linking. Spotify also offers widgets, which allow embedding players and playlists in websites, however, the widgets only work by directly linking to functionalities within a Spotify app. Several other music streaming services offer more possibilities in incorporating music playback in new environments, but, like Spotify, these all require a user to be logged in to the service. Conclusively, as a web technology, Spotify opens up no core functionalities to developers, nonetheless it does offer metadata of their music library and easy linking to in-app functionalities.

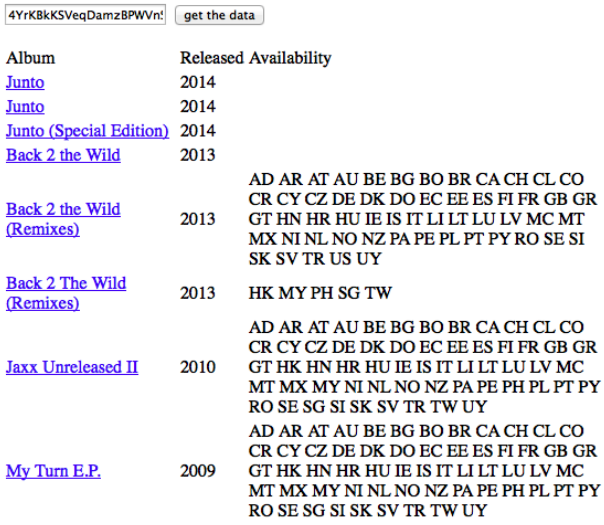

Figure 3. Screenshot of the lookup page

REFERENCES

1. BeatsMusic - Getting Started. https://developer.beatsmusic.com/docs/read/Home.
2. Blixt, A., and R.V. Santos. *Spartify.* http://www.spartify.com/.
3. Bradner, S. The Internet Standards Process -- Revision 3. *Harvard University*, October 1996. http://tools.ietf.org/html/rfc2026#section-2.2.
4. Closure of Spotify Apps Submission. *Spotify.com.* https://developer.spotify.com/news-stories/2014/03/24/closure-of-spotify-apps-submissions/.
5. Covify: Scanning your albums into Spotify. http://beta.covify.com/.
6. Deezer For Developers. *Deezer.com.* http://developers.deezer.com/login?redirect=/api.
7. Deezer for Developers: Javascript SDK. *Deezer.com.* http://developers.deezer.com/sdk/javascript.
8. Developers/Documentation. *Rdio.com.* http://www.rdio.com/developers/docs/web-playback/index/.
9. Developers: Getting Started. *Beatsmusic.com.* https://developer.beatsmusic.com/docs/read/web_playback_api/Getting_Started.
10. Documentation - Rdio Developers. *Rdio.com.* http://www.rdio.com/developers/docs/web-service/overview/.
11. Download Spotify. *Spotify.com.* https://www.spotify.com/us/download/.
12. Forgotify: Discover a previously unheard Spotify track. http://forgotify.com/.
13. Geere, D., Spotify hits 10 million users and 10 million tracks. *Wired.co.uk* (September, 2010). http://www.wired.co.uk/news/archive/2010-09/15/spotify-milestones/.
14. How long can I cache offline content? *Spotify.com.* https://support.spotify.com/us/learn-more/faq/#!/article/How-long-can-I-cache-offline-content.
15. Information - Spotify Press. *Spotify.com.* http://press.spotify.com/nl/information/.
16. Kahn, T., Spotify Random Song Generator. http://www.karnhuset.net/demos/spotify/randomSong/.
17. Linking to Spotify. *Spotify Blog. Spotify.com* (January, 2008). https://news.spotify.com/nl/2008/01/14/linking-to-spotify/.
18. Listen Offline. *Spotify.com.* https://support.spotify.com/us/learn-more/guides/#!/article/Listen-offline.

19. Metadata API - Developer. *Spotify.com.* https://developer.spotify.com/technologies/web-api/.
20. NME on Spotify. http://www.nme.com/spotify.
21. Peckham, Matt. 13 Music Streaming Services Compared by Price, Quality, Catalog, Size and More. *Time Magazine*. March 19, 2014. Print. http://time.com/30081/13-streaming-music-services-compared-by-price-quality-catalog-size-and-more/.
22. Search - Developer. *Spotify.com.* https://developer.spotify.com/technologies/web-api/search/.
23. Songkick Concerts. http://open.spotify.com/app/songkickconcerts.
24. Spotify: A Perfect Platform for Apps. *Spotify.com* (November, 2011). http://press.spotify.com/us/2011/11/30/spotify-a-perfect-platform-for-apps/.
25. Spotify Apps API Reference Guides. *Spotify.com.* https://developer.spotify.com/technologies/apps/reference.
26. Spotify Background Information. *Spotify.com.* https://d2us6zencw9qvn.cloudfront.net/wp/u/spotify-background-information.pdf.
27. Spotify hits 10 million global subscribers. *Spotify.com* (May, 2014). http://press.spotify.com/us/2014/05/21/spotify-hits-10-million-global-subscribers/.
28. Spotify inside your iOS app. *Spotify.com* (August, 2011). https://developer.spotify.com/news-stories/2011/08/31/spotify-inside-your-ios-app/.
29. Spotify Music Catalog - Search Song Library. http://www.linein.org/examples/spotify/.
30. Spotify's Progress So Far. *Spotify.com.* http://www.spotifyartists.com/spotify-explained/#spotifys-progress-so-far.
31. Spotify Widgets - Developer. *Spotify.com.* https://developer.spotify.com/technologies/widgets/.
32. Svensson, G., Spotify Search Add-on. https://addons.mozilla.org/en-US/firefox/addon/spotify-search/.
33. Uniform Resource Identifier (URI) Scheme Name: Spotify (2012). https://www.iana.org/assignments/uri-schemes/prov/spotify.
34. We've only just begun!. *Spotify Blog. Spotify.com* (October, 2008). http://news.spotify.com/uk/2008/10/07/weve-only-just-begun/.