ECE/CS 476 Final Report
DeCentralized Playlist
Winter 2019

Martin Barker
School of Computer Science
Oregon State University
Corvallis, OR 97331
barkemar@oregonstate.edu
www.martinbarker.me

***Abstract*—DeCentralizedPlaylist provides the framework for collecting a user's listening history across multiple platforms and devices. Listening data can be collected from various sources including file players, Spotify, Soundcloud, and YouTube in order to maintain a concurrent list of every song that a user has listened to. With this cross-platform listening history stored in a central database, it gives the opportunity to generate more accurate analysis and better recommendations.**

1. Introduction

Music is a stable industry. Not stable in the sense that it's immune to disruptive change, but stable in the sense that it will always exist. People are always going to be listening to music, which means that relevant innovations in technology will usually impact the music industry in some way. In today's world it's become expected for companies to use collected data in order to better advertise products, and music is no different. With the rise of streaming services such as Spotify, YouTube, Soundcloud, Apple Music and more, a new market for analysis-based music recommendations has been created.
Once a large enough data set of someone's listening history has been acquired, technology such as machine learning can be implemented in order to generate music recommendations and suggestions. Once a platform can make well-educated, accurate suggestions, they have a better chance of introducing an artist to a listener who may financially support that artist in the future. To generate better analysis however, you need better data, and the current state of data collection for listening history is isolated and restrictive.

2. Problem Statement

In order to generate the best possible analysis of a person's listening history, you need to have data which gives an accurate history of all the music that person has listened to. The currently available music services however, only offer analysis to music consumed on that platform, as is in the best interest of the company in order to retain users. As these platforms grow and develop, they offer more specific and unique services in order to differentiate from the competition. Resulting in a music ecosystem where each platform targets a different audience, and specific communities thrive on each service.
With each music platform specializing in different communities, their available analysis will be limited, as it doesn't take into account what music their user listens to on rival platforms. While a platform like Spotify will offer analysis and recommendations based on music consumed inside their software, their isolated analysis will not take into account what music the user listens to on a different platform such as Soundcloud. Which will lead to Spotify's music analysis having a limited scope.
The internet's evolving copyright protection systems demonstrate this point well, as platforms such as YouTube have become a wealth of old, rare records which have never been entered into the modern music streaming ecosystem, and haven't seen physical pressings in decades. There are plenty of people who consume music exclusively on one platform such as Spotify in order to get the newest releases, and for these people the DeCentralizedPlaylist is not needed. The target audience for this product are people who listen to music through multiple platforms (Spotify, Soundcloud, YouTube, Bandcamp, Files) across separate devices (Desktop, Laptop, Android Phone), and the goal is to give these user's the accessible analysis and consistent listening history that one would receive from using a single platform exclusively.

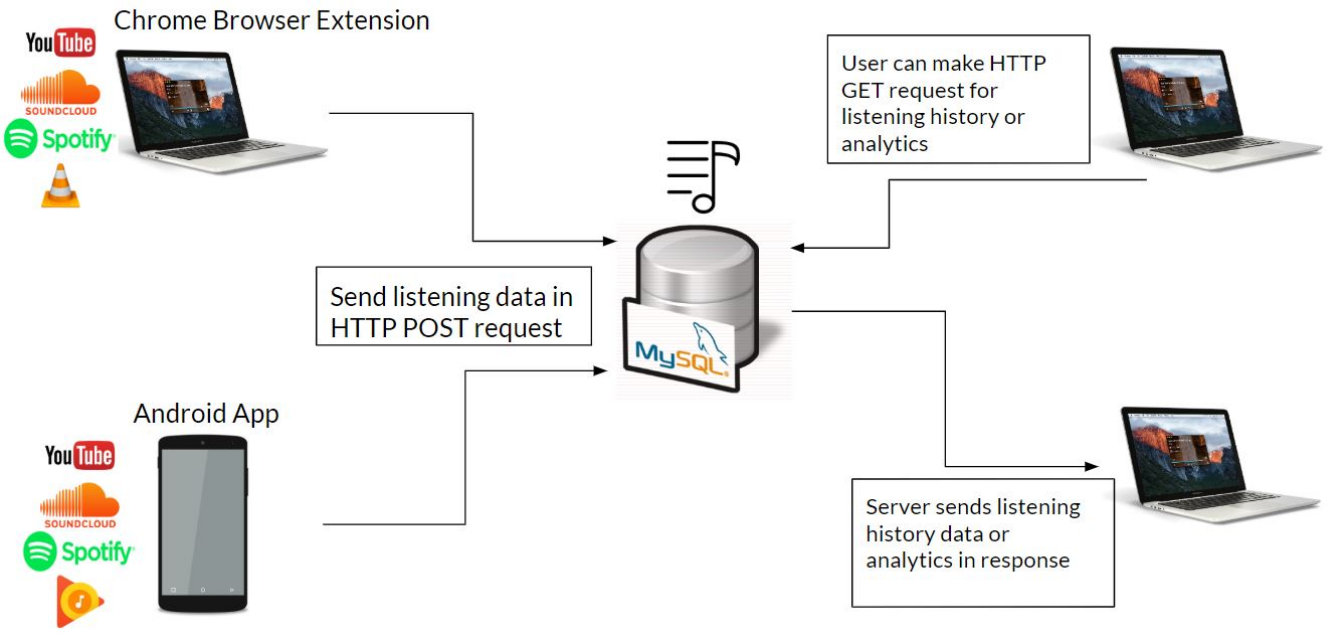
3. Solution Approach

In this section we are going to look at the implementation of this system, which is comprised of three separate entities:

- Android Application
- Chrome Extension
- MySQL Webserver

3.1. Topology

The following image shows an overview of communication between the Android App, Chrome Extension, and MySQL server:



3.2. Assumptions

In order to achieve results for this project on a shorter timeframe, the main emphasis of my work was done on creating the underlying network functionality required for the DeCentralizedPlaylist to work. This included creating the Android app and Chrome extension, getting them to collect data correctly, performing successful HTTP Post requests in order to send the data to the MySQL server, and writing the PHP scripts which parse and insert the data into the correct database table.
I also decided early on what metadata information was required for this project to work in it's most basic implementation, these fields include Title, Artist, Album, DateTime, Source, and Device. The fields pertaining to actual music metadata (Title, Album, Artist) are the most commonly occurring metadata across mobile listening platforms.

3.3. Android Application

This project initially started with the android app, written in Java, after I realized that no matter what source of audio you listen to, you can always find that audio's media metadata from the Android notification dropdown menu. This remains true for both music streaming sources such as YouTube, Spotify, and Soundcloud, as well as file playing source such as Blackplayer and Google Play.
To implement the data collection aspect of this app, I made a very simple interface with a button for enabling / disabling the collection of data. Once enabled, the app will begin to listen for when the UpdateMediaMetadata API function is called. Once this function is called, the app will try to store as much information about the newly playing song as possible, such as Title, Artist, and Album.
Out of the sources I tested on the app was able to pick up on these three tags in Spotify, Soundcloud, and file playing applications. In instances such as with YouTube or Chrome mobile web browser audio sources, the only field recorded would be the Title, while Artist and Album are set to null. This limited collection of data will addressed later on in this report, in the section covering results. The API calls for this app in it's current implementation require at a minimum an Android 4.4 KitKat (2013) operating system.
Once the metadata for a song has been collected, it is appended to a 'songsArray' stored in the app's local storage on the phone. When the user presses a button in the interface, this songsArray is retrieved and converted to JSON data, with login credentials appended to the start.

Example JSON file:


```
{
    "Username":"martinbarker1",
    "Song1":{
        "Title":"Norway",
        "Artist":"Beach House",
        "Album":"Teen Dream",
        "DateTime":"2019-3-9 13:17:17",
        "Source":"blackplayer",
        "Device":"androidPhone1"
    },
    "Song2":{
        "Title":"Golden Symmetry",
        "Artist":"Von Haze",
        "Album":"Kar Dee Akk Ake",
        "DateTime":"2019-3-10 13:17:17",
        "Source":"spotify",
        "Device":"browser1"
    },
    "Song3":{
        //...etc
    }
}
```

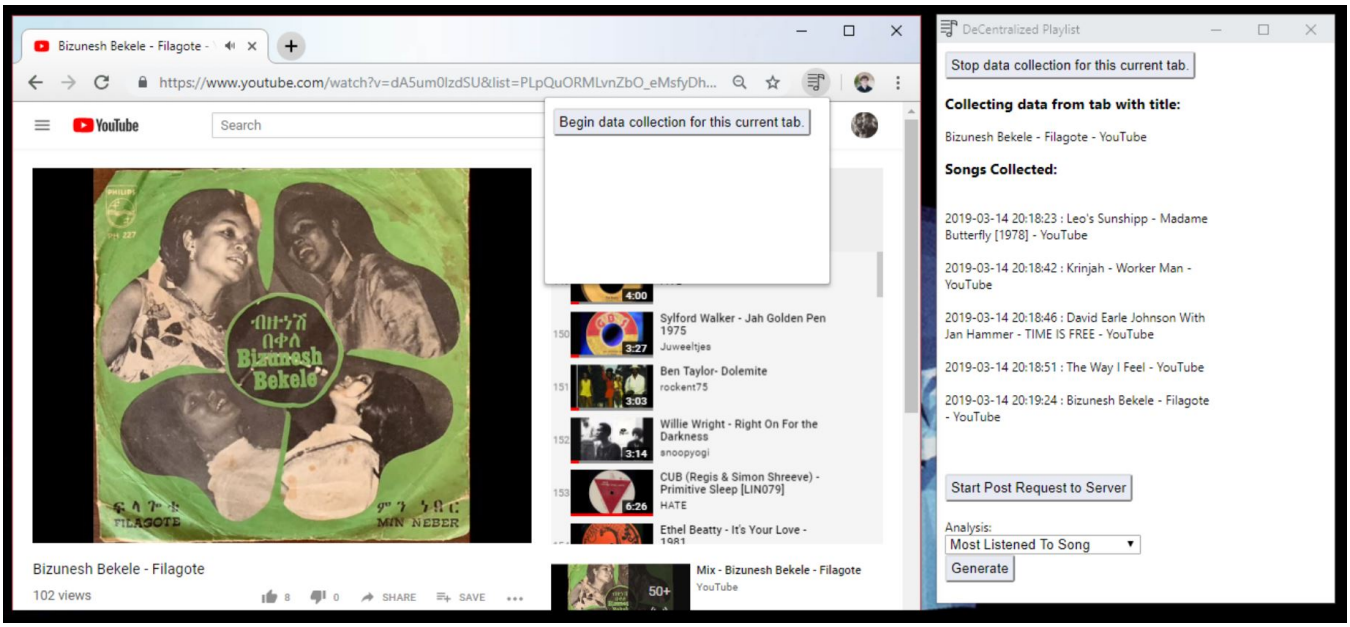

Currently implemented Android App (placeholder UI):

When the user specifies, this JSON data will be sent to the URL ”www.decentralizedplaylist.com/receiveSong.php”

3.4. Google Chrome Extension

The Chrome extension, written in Javascript / CSS / HTML, takes a different approach to detecting and storing a user’s listening data by analyzing the title of a selected tab. From my findings you can get a good amount of data from the tab title alone, which usually consists of concatenations containing media metadata such as an artist name, release name, and song title. In some cases this information is uniformly formatted, such as Spotify which will always display the currently playing media in a Chrome browser through the tab title as ”Title - Artist”.
In other cases, such as YouTube, there is no uniform metadata standard, and people often times title music videos in different, strangely formatted ways. After a user clicks on the Chrome extension to open it, they are greeted with a button labeled ”Begin data collection for this current tab”. If the user clicks this button, it will detect and store their current targetTabID in Chrome’s remote sync storage. The extension will then open a popup window which will remain open even after the tab refreshes / changes pages. This popup window will display the current title of your targetTabID, as well as display all songs which have been collected while your had the popup window open.
The extension uses Chrome’s javascript API to detect when the tab’s title changes, as well as to detect and store the targetTabID. Each song that is detected gets stored in an array stored in Chrome’s remote sync history. Once the user presses the ”Send Post Request to Server” button, this array will be formatted into a JSON data type identical to the one created in the Android Application. Then the Chrome extension will establish a Cross-Origin Resource Sharing (CORS) HTTP Post request to the URL ”www.decentralizedplaylist.com/receiveSong.php”.

Currently implemented Chrome Extension:



3.5. MySQL Server

When the file 'receiveSong.php' is called in an HTTP Post request, the first thing it will do is store the json data in a string by calling file_get_contents('php://input');
At this point the string will still be URL encoded, meaning that characters such as spaces and punctuation have been replaced with characters such as %20
Before the JSON data can be interpreted, the string needs to be urldecoded so that the replaced characters return to normal. After this step the string is converted to a JSON object with a call to the PHP function json_decode(). Now that the PHP file has a valid JSON object, it can iterate through each "Song" element and create an SQL insert query for the JSON data.
In the current implementation, the PHP file just inserts this data into the user's SQL Table without any further analysis. An example of the achieved data can be seen in in the following Performance Evaluation section.

4. Performance Evaluation

Example database values resulting from analysis received from the Android Application and Chrome Extension

ID	DateTime	Title	Artist	Album	Source	Device
2	2019-03-10 13:17:17	Golden Symmetry	Von Haze	Kar Dee Akk Ake	blackplayer	Android
3	2019-03-10 13:17:21	Breed	Nirvana	Nevermind	Spotify	Android
4	2019-03-10 13:17:34	Golden Symmetry	Von Haze	Kar Dee Akk Ake	Soundcloud	Android
5	2019-03-09 13:17:40	Norway - Beach House			Spotify	Chrome
6	2019-03-14 19:52:39	21st CENTURY - REMEMBER THE RAIN - YouTube			YouTube	Chrome

4.1. Parameters to evaluate effectiveness of solution

The main parameter we will be using to evaluate our resulting listening history database table will be the accuracy of data collected. Ideally we want the table to be as accurate of a representation of someone's listening history as possible. For this goal we need as many data fields as possible, and develop solutions to evaluate rows for missing data. One way to collect large amounts of data for evaluation is to use the DeCentralizedPlaylist applications all day in order to cover my cross-platform listening habits. Once data from a day is collected, we can evaluate each collected row and determine whether or not it's inclusion in my listening history is warranted.

4.2. Results

From our generated results, we can see that songs captured with the Chrome extension will have limited data compared to songs listened to on mobile devices. Another possibility not represented in the database table image is the capability for the android app / Chrome extension to capture non-music listening data accidentally. Since the Android application and Chrome extension, in their current implementation, just read changes from a specified metadata source, it's possible for non-music audio sources such as phone calls and non-music YouTube videos to be accidentally recognized as a song.

4.2.1. Possible solutions to non-music captured data. One solution for the problem of accidentally capturing non-music sources is giving the user the ability to edit and delete their collected listening data before it is sent to the server. This would require using CSS for the Chrome extension and XML for the Android Application to display the data in a user friendly way. Having a user review their listening data would require a small amount of time, but would prevent mistakenly recorded audio sources from being inserted.
Another opportunity to fix or edit already inserted data rows would be from the Chrome extension data retrieval POST information. With a user's database listening history retrieved and displayed neatly in the Chrome extension, the user could manually edit specific rows to include more data, or delete entire rows all together if they think it doesn't represent their listening history accurately.

4.2.2. Possible solutions to rows with missing data. Some entries such as those sources from YouTube on the Chrome extension will have limited data fields, with the video name being stored in the title, while the artist and album fields are null. One way to get more metadata for a song using only it's YouTube video title is to use the Discogs API. This API can be implemented in the same PHP which receives and inserts the JSON data. Rows parsed from YouTube will already have 'YouTube' set in their 'Source' column, and usually there will be enough data in the title alone to find the release on Discogs by performing a simple search using the title string. In order to mitigate against identifying the wrong release, which is a real possibility for titles with less unique information, each Discogs API search which is performed can be assigned a value to represent the scripts 'Degree of Certainty'. This Degree of Certainty would represent how confident the script is that it has recognized the correct release out of all the possible results. The implementation of this system are complicated and would deserve an entire paper on their own, so for now we will make the assumption that YouTube video titles have enough unique metadata for the Discogs API to identify the correct release.
In some cases however, the Discogs API won't be much help for identifying a release from just the YouTube title string. As there is a rising number of official YouTube music uploads that only include a song's title in the video title, and don't include any other information such as artist name or year. This makes identifying songs with simple titles such as "LIFE" almost impossible to do with just using the Discogs API alone. In this case, we would need to implement the YouTube PHP API in a similar fashion. Similar to how the Discogs API implementation was proposed above, the YouTube API would intercept data before it is inserted into the user's table, and try to get as much additional music metadata information on a row as possible. This could be achieved by appending a YouTube video's URL to the 'Source' column, which the YouTube API could use to make a request for the video's channel name and description. With the channel name and description parsed, these sources can be used in different concatenations with the video title for a Discogs API search in order to try and find the correct release.

Ultimately the goal for using these API's to find additional information on a release would be to identify a song's correct Discogs URL listing. Once this is achieved, additional data can be parsed such as a song's genre / style / year released. This data would be very beneficial for generating more in-depth analysis such as analyzing how a user's tastes have changed over long periods of time.

5. Conclusion

In working on this project, I learned about Android App development, Google Chrome extension development, and sending / receiving JSON data over HTTP requests. These are all things which I had no prior experience in, and took as a challenge to learn in a couple weeks for completing the groundwork implementation of the DeCentralizedPlaylist. While what I achieved doesn't have the data accuracy that something like this would need to thrive and compete with analysis available on other music platforms, I was still happy to get the connection aspects working as desired. Developing on Android was challenging initially, but even when I faced roadblocks I knew it was worth it to continue because I was working on an idea I felt passionate about.

5.1. Future Plans

I plan to continue working on this project, and eventually run a small closed beta test for users who sign up with their email at www.decentralizedplaylist.com. I want to complete the following items in roughly this order:

- Better data collection on Android
- Username / password login credentials and support for multiple users
- Protection against SQL injections
- Give user's the ability to edit / delete data before it gets sent to the server
- Implement the Discogs PHP API to intercept data before being inserted and to find more metadata for a row.
- Implement the YouTube PHP API to intercept data before being inserted and to find more metadata for a row.
- Add better and more analysis options
- Move the database format over from one table to two tables, and set up foreign key relationship with a new 'Songs' table.
- IOS Application Support
- Add non-mobile file player analysis in the Chrome extension