Database project 2
My Music Hub

Pei-Lun Liao N18410090 pll273@nyu.edu
Chia-Hsian Lin N17505647 chl566@nyu.edu

In the second project, you have to create a web-based user interface for the database designed in the first project. In particular, users should be able to sign up, create a profile, log in, play tracks, make playlists, like an artist, rate songs, follow other users, and browse and search for music. After login in, there should also be a welcome/home page That displays recent events that are relevant to the user, e.g., new track by artists they like, new playlists from people they follow or containing artists they like, etc. (You can decide what would be interesting to display here.)

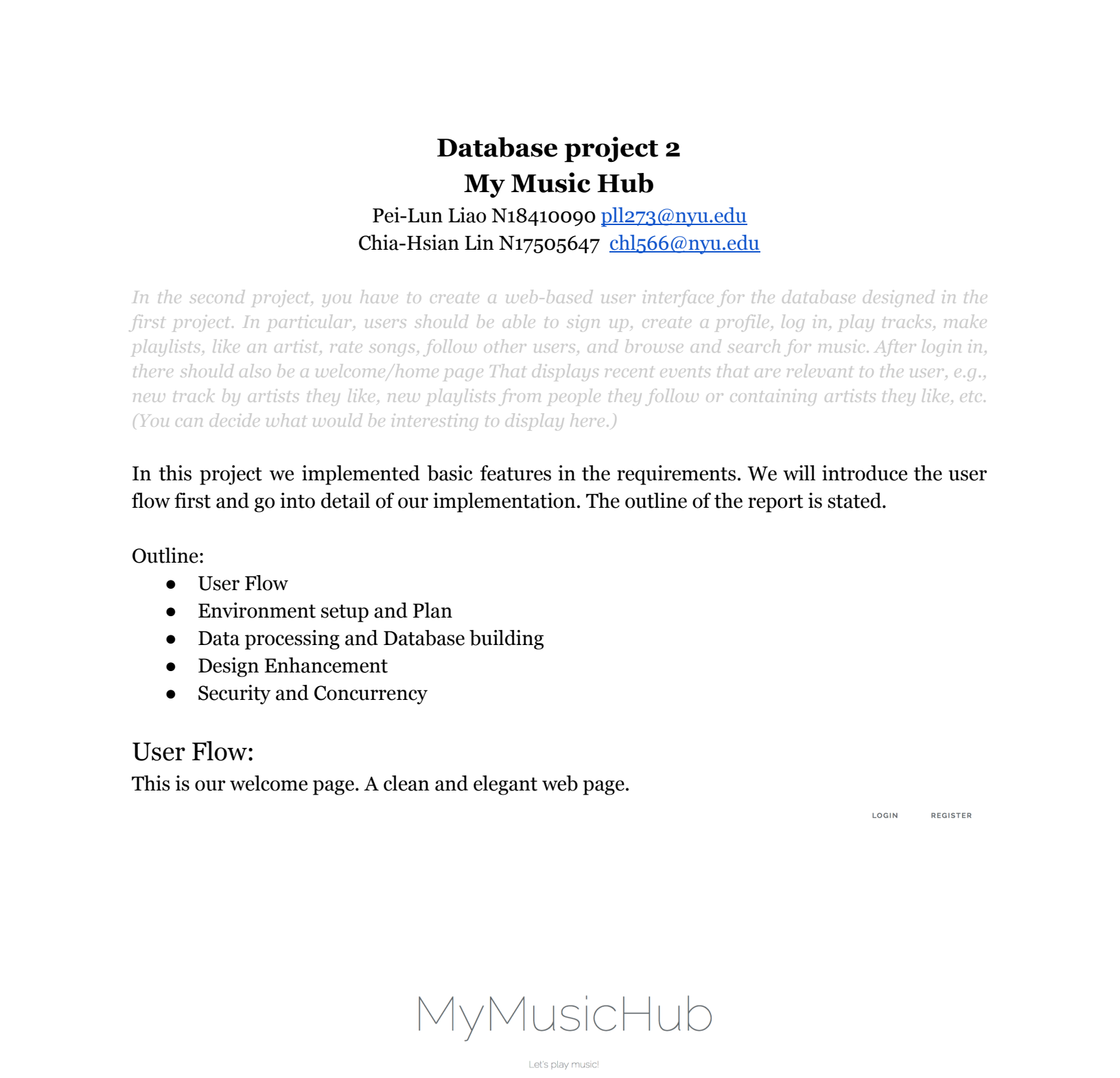
In this project we implemented basic features in the requirements. We will introduce the user flow first and go into detail of our implementation. The outline of the report is stated.

Outline:

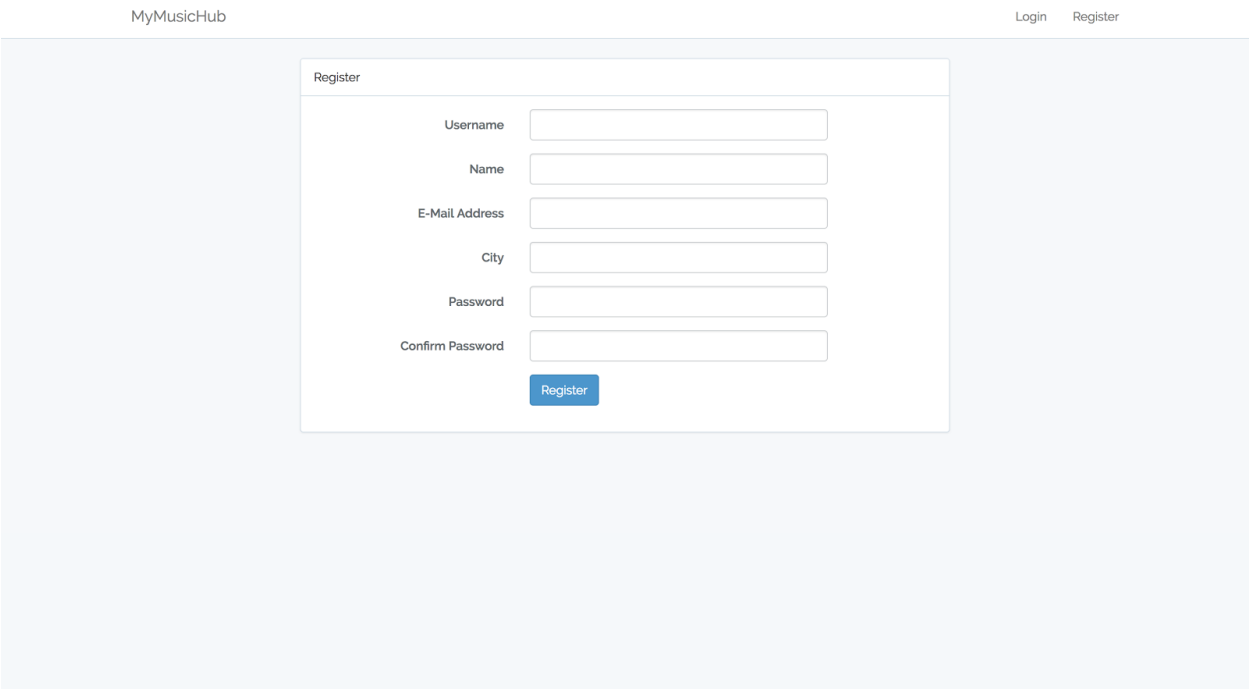
- User Flow
- Environment setup and Plan
- Data processing and Database building
- Design Enhancement
- Security and Concurrency

User Flow:

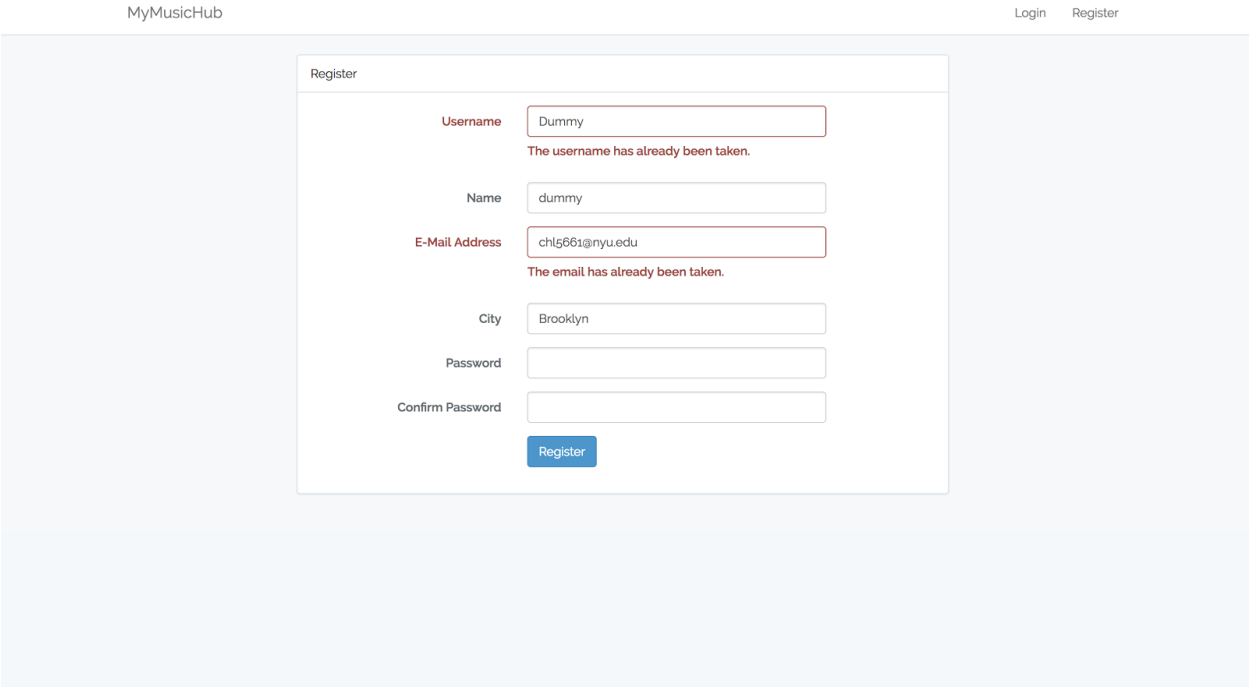
This is our welcome page. A clean and elegant web page.



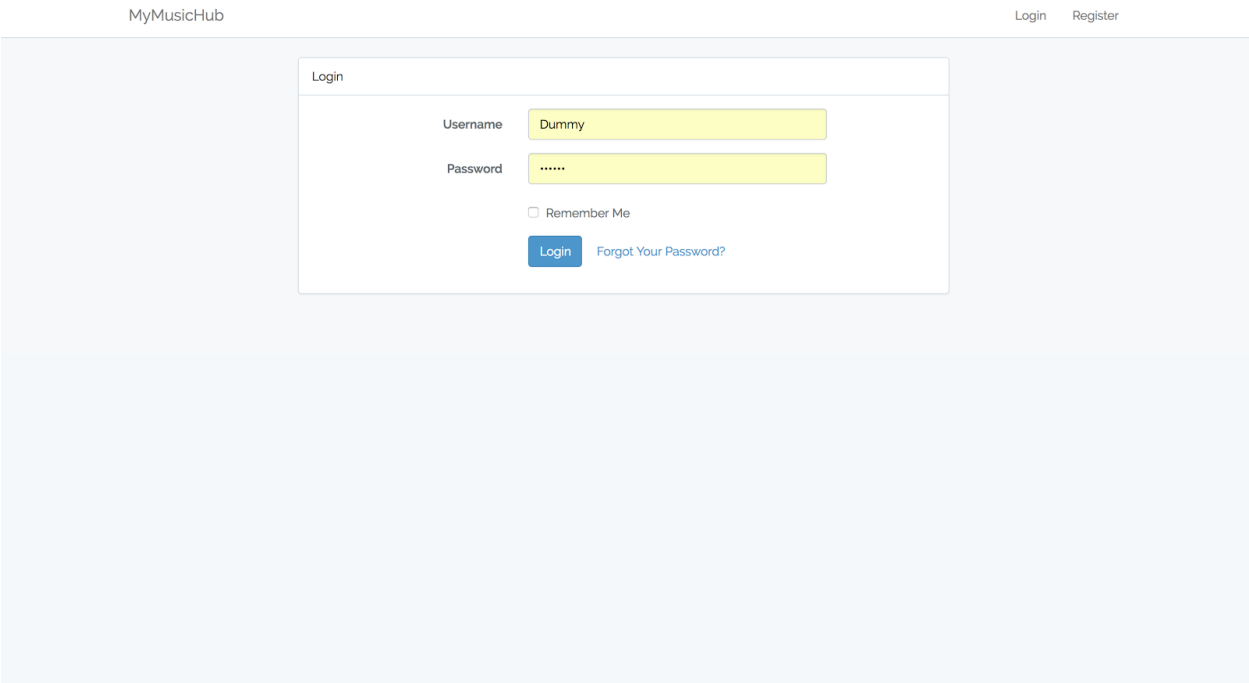
User can sign up our webpage by clicking the button on top right of the page.



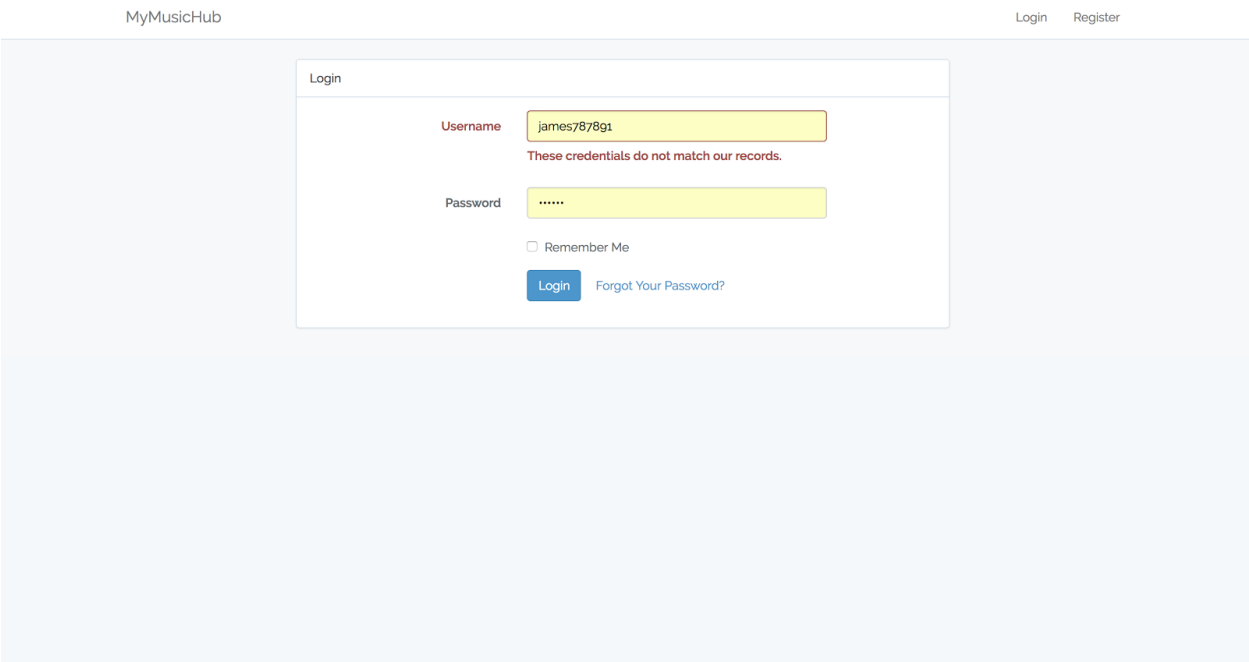
The username is the user account. The user is not allowed to register an existing account. Besides,the email has to be unique and in the valid email format such as xxx@mail.com.



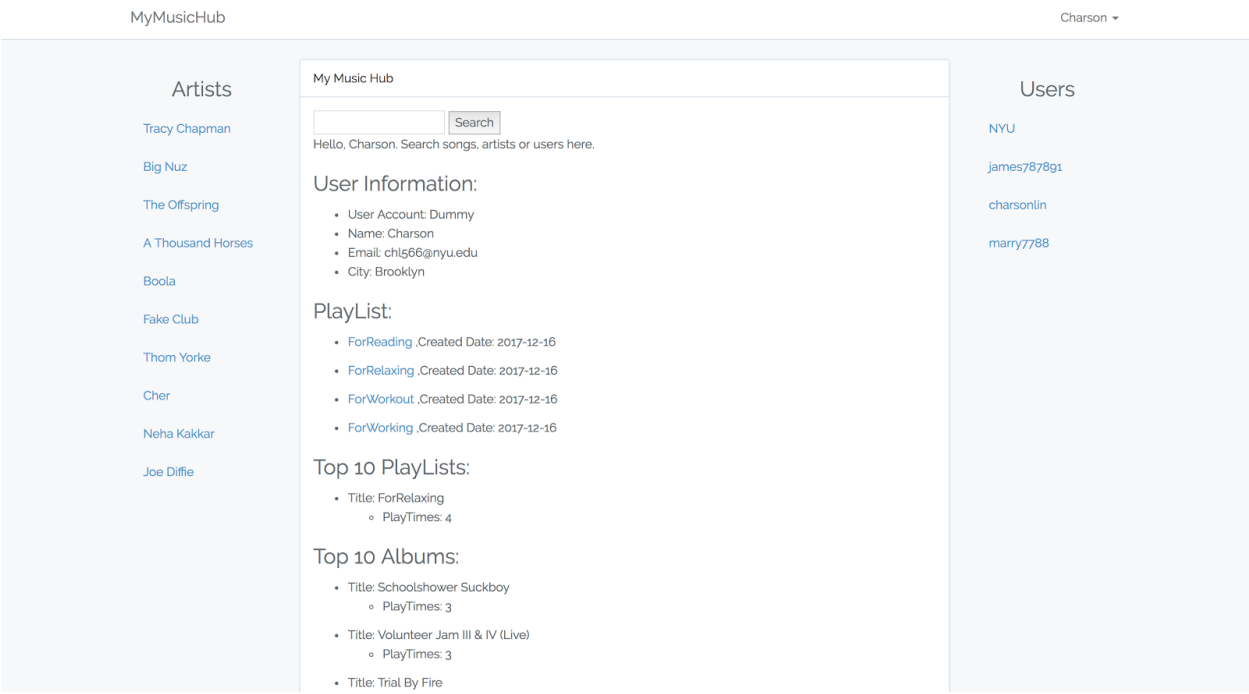
For the existing users, they can log into our page by clicking Login button.



We also handled the case that the user doesn't exist or login with wrong password.



After logged into the page a welcome dashboard is displayed to the user.



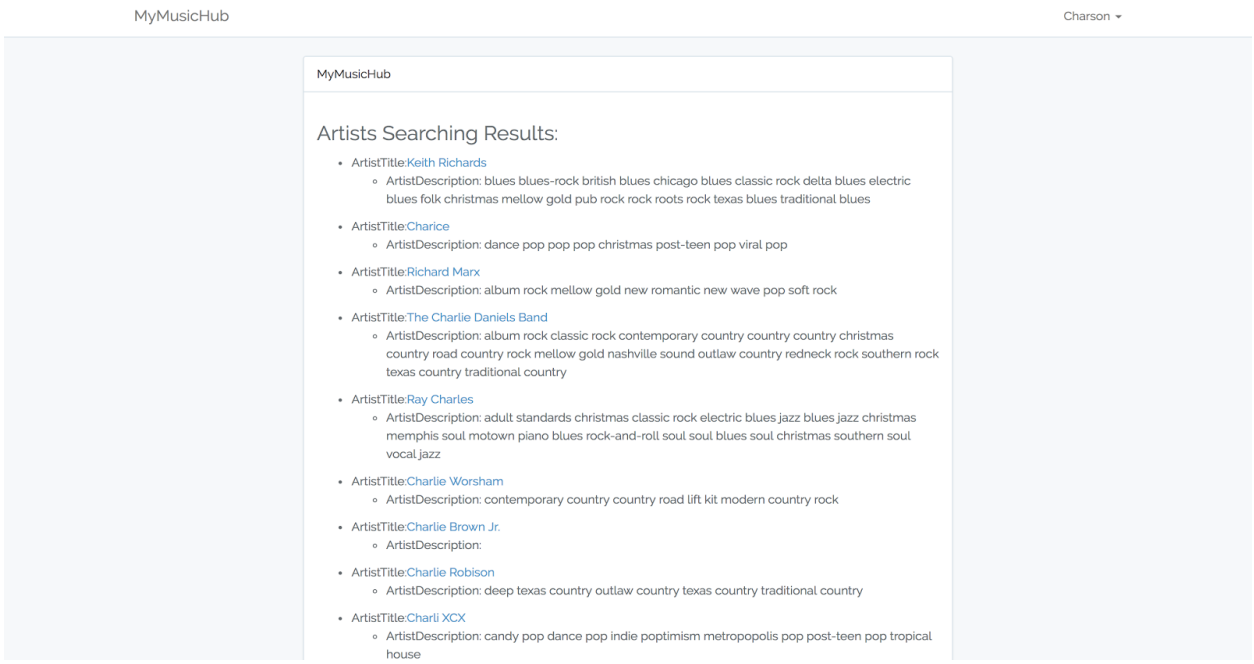
As you can see on this page, on the left-hand side, we show the artists with our ranking setting. The artists liked by the user will be put on the top of the list and the popular artists will follow the user-liked artists. The popular artists is defined by the number of like he or she received.

In the middle, basic user information is displayed. The PlayList is the playlists created by the particular user. As you can see, we provide the suggestion of the top hits of playlist and albums for user to search.

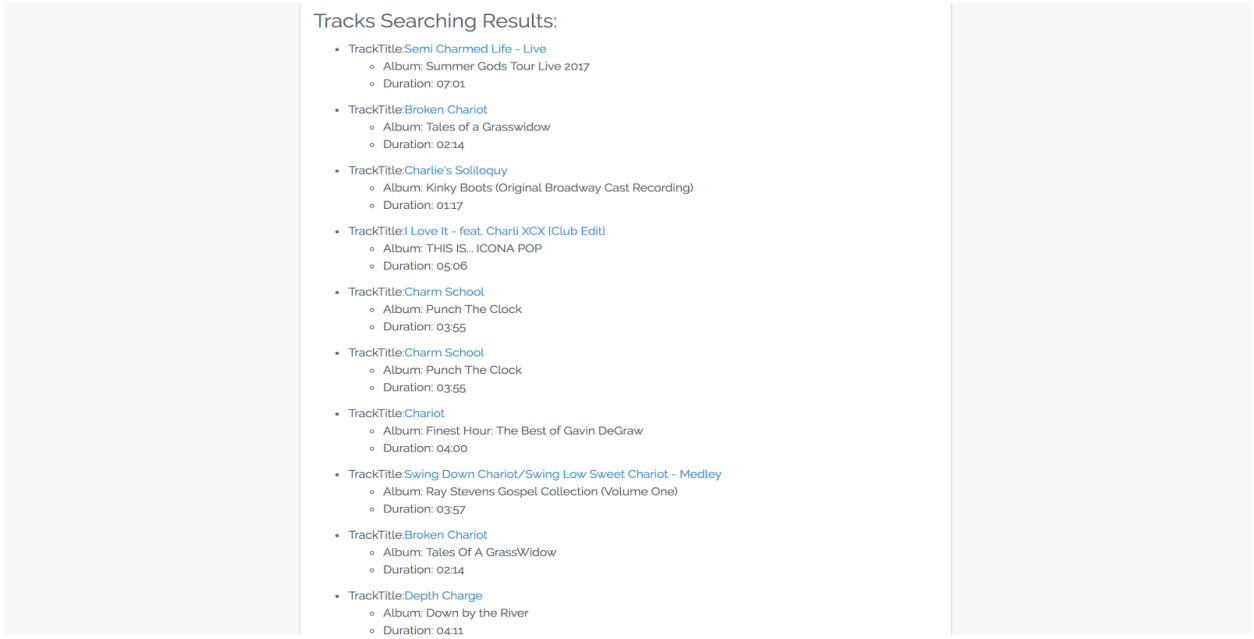
On the right-hand side, we list the users followed by the particular user and the popular users. The ranking algorithm is same with the one we use to rank artists.

On the top of the page, user can click our brand name, MyMusicHub, to go back to dashboard. Also, there is a navigation button on the top right of the page for user to browse our web page.

Also, we provide the function to search tracks, artists, and users by keyword.

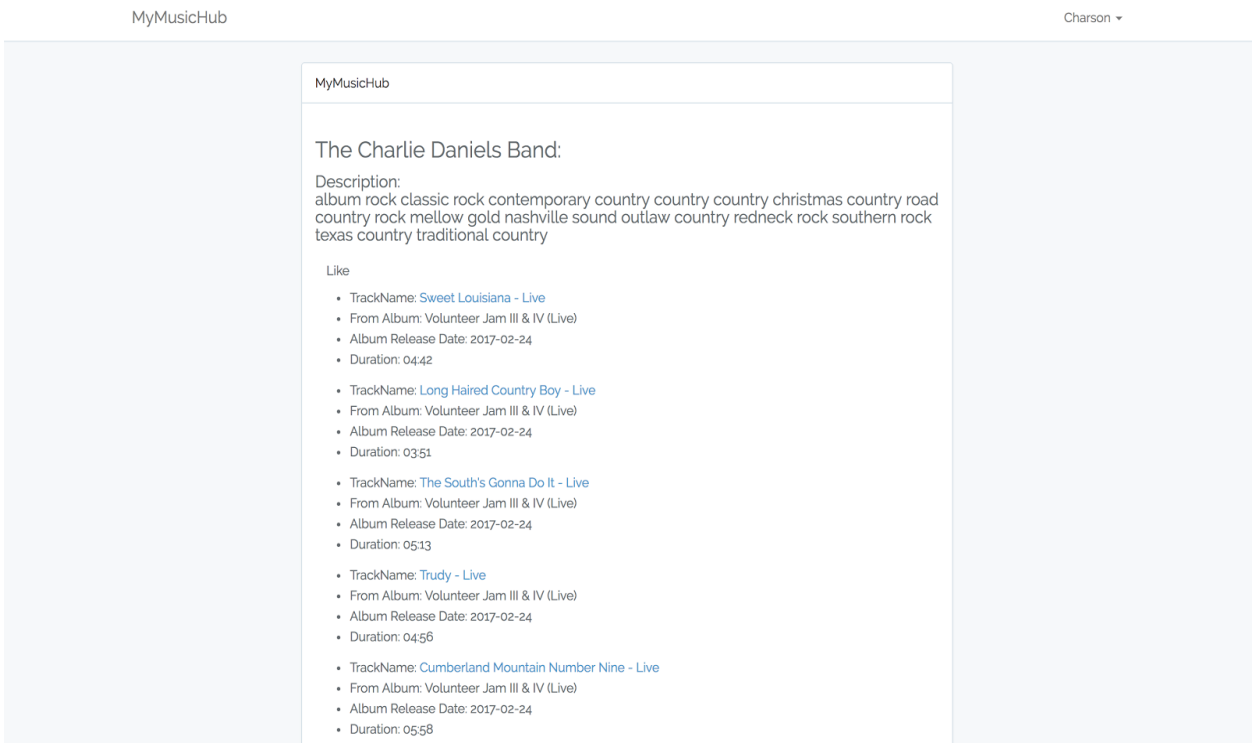


This is the searching results of artists by the keyword ‘Char’. We search our database using the keyword ‘Char ’to do fuzzy string searching and matching to find the particular keyword appears in the artists’ names, tracks’ titles, or user’s usernames.
This is the searching results of Tracks by the keyword ‘Char’.

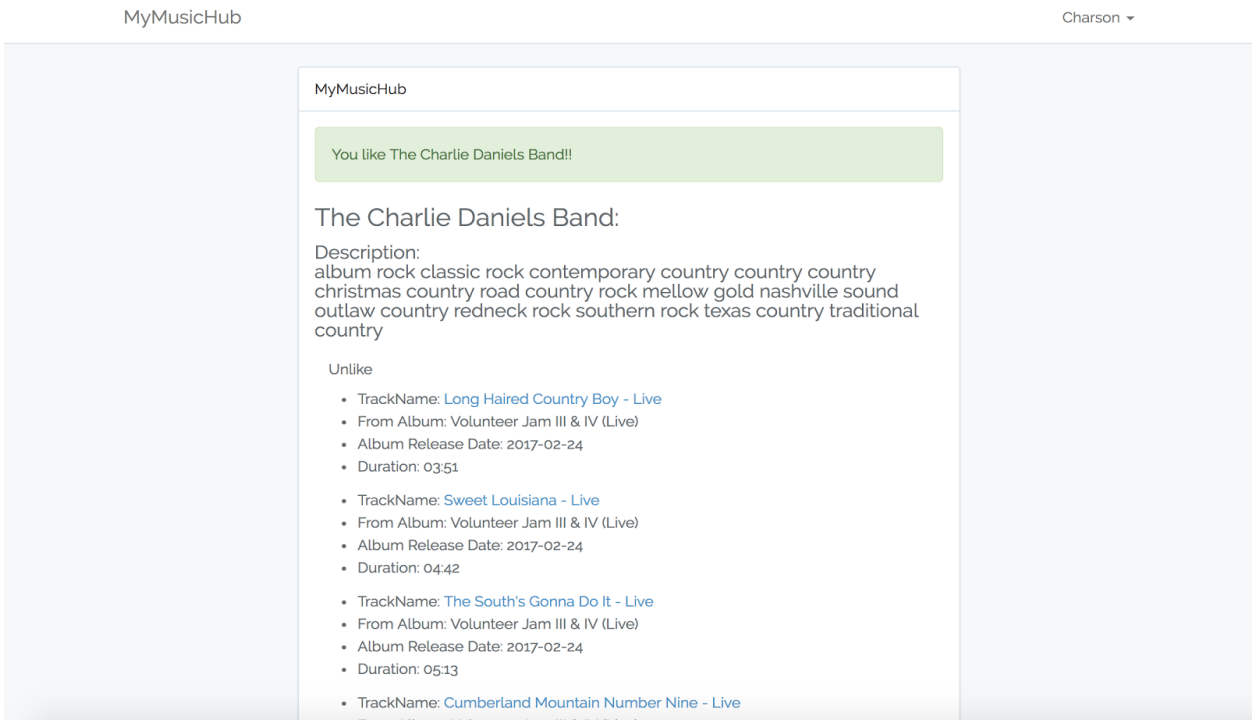


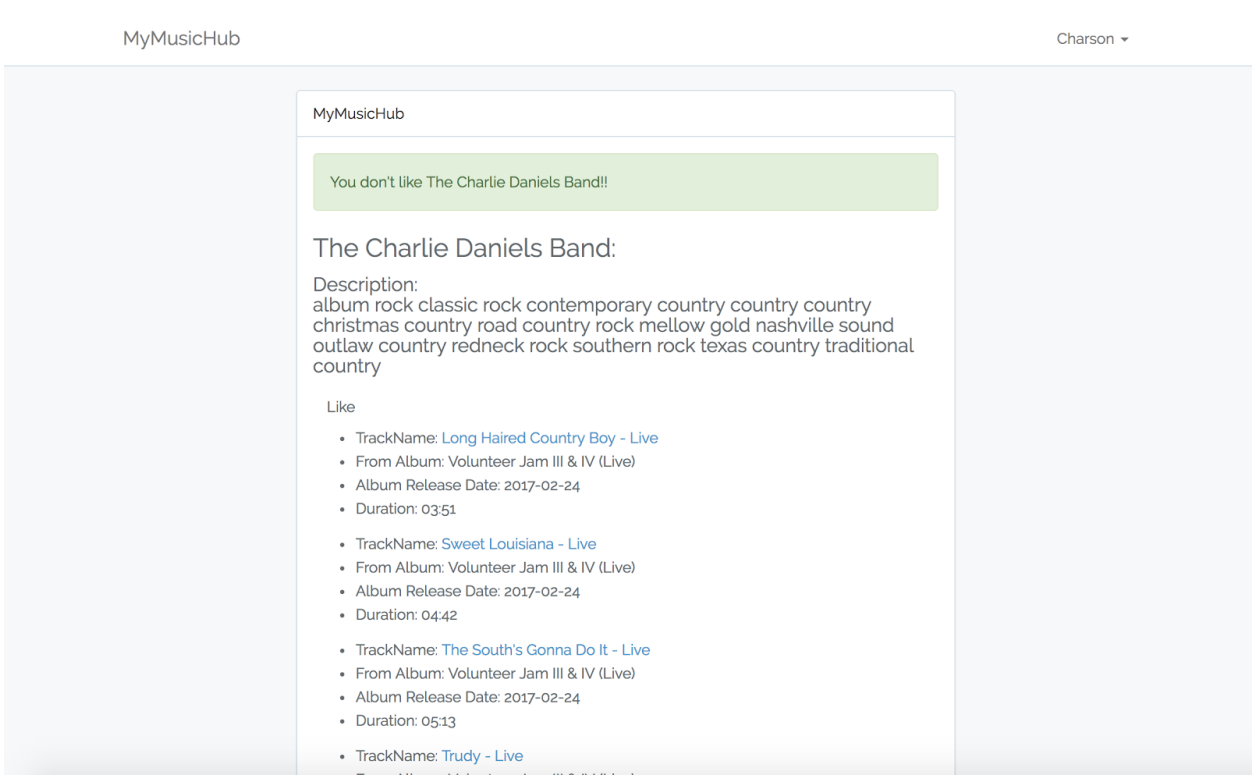
This is the searching results of Users by the keyword ‘Char’.

After clicking any artist on the searching results page. There is a bunch of information regarding to the particular artist. The albums and tracks created by the artist are listed on the page.

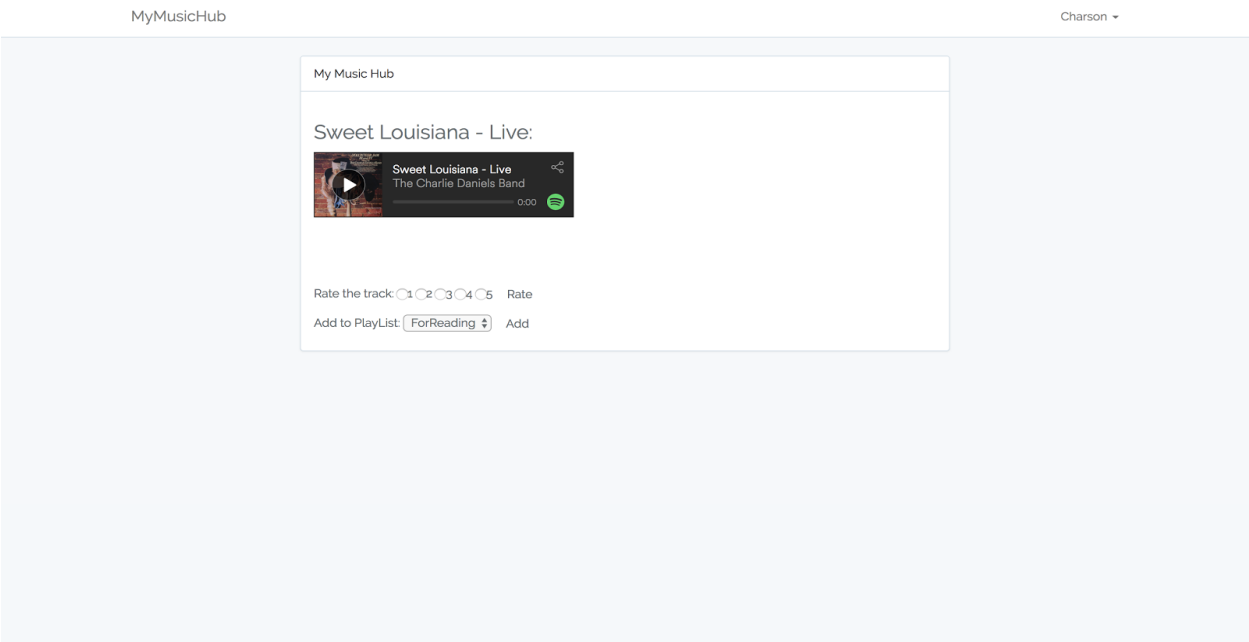


The user is allowed to like or unlike the artist by clicking the ‘like’ or ‘unlike’ button on the artist page.

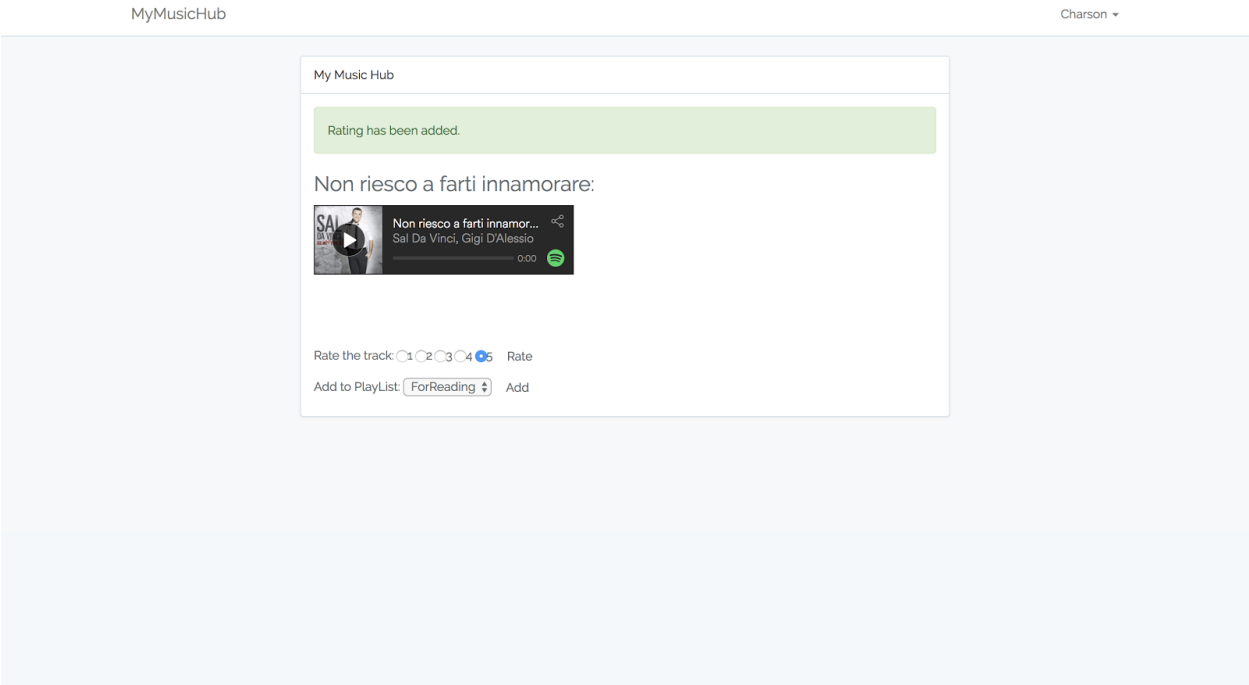




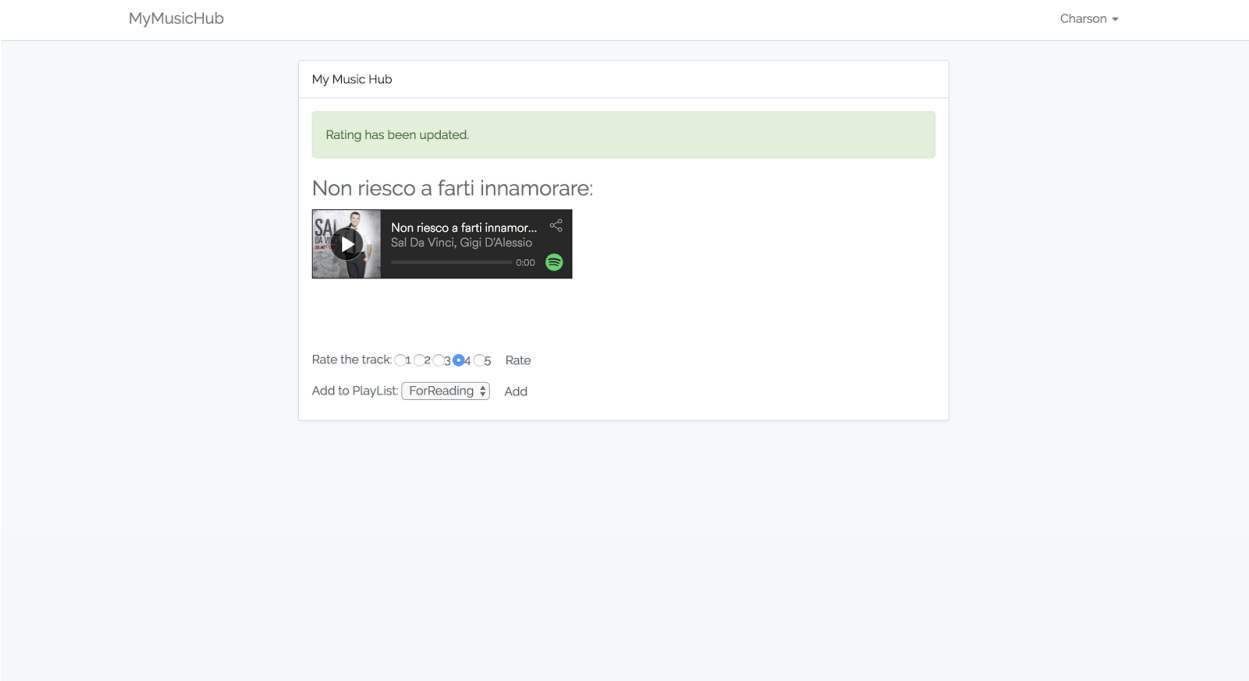
The user could click the track to play the music. At the time the user clicks on the track url, our system will record the user action, when and who plays which particular track, into the PlaybyAlbum table. The user can play the music with the Spotify embedded Web player.



User is also allowed to rate any track as he wants from point 1 - 5. After clicking the rate button, the rating record will be stored into database.

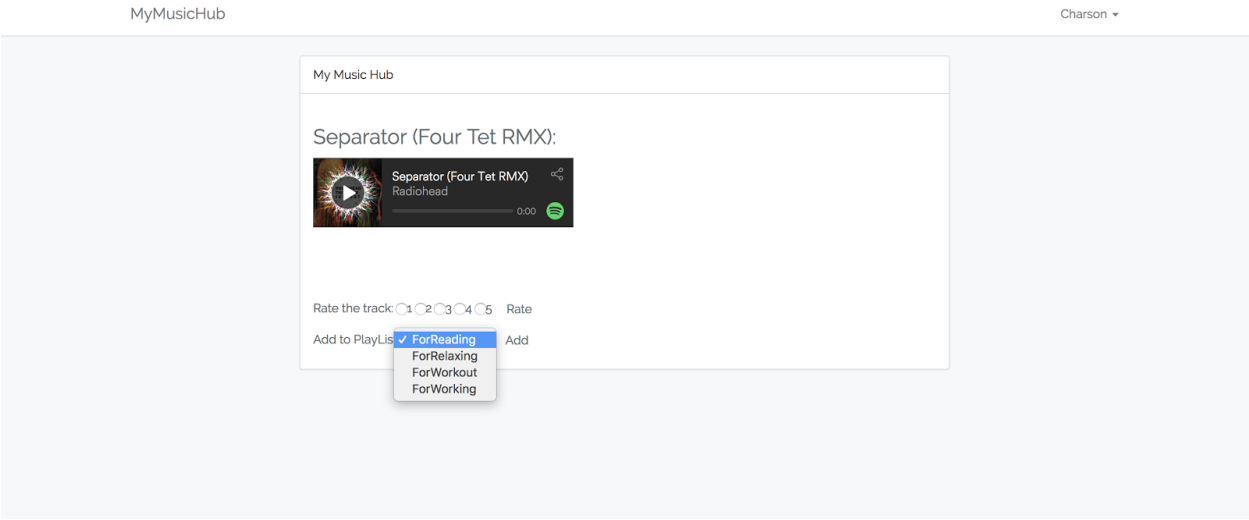


The user is allowed to rerate the track. Once after rerating the track, the page will reveal that the rating has been updated.



Once the user onclick any song, he will able to play the song, rate the song and add the song by selecting the playlist which he created. At the time the user gets into this page, there is a playing record stored into the table PlaybyAlbum.

The user could scroll down the Playlist to choose to which playlist he want to add the particular track.

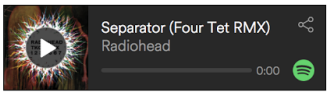


MyMusicHub Charson

My Music Hub

Track has been added.

Separator (Four Tet RMX):



Rate the track: 1 2 3 4 5 Rate

Add to PlayList: ForReading Add

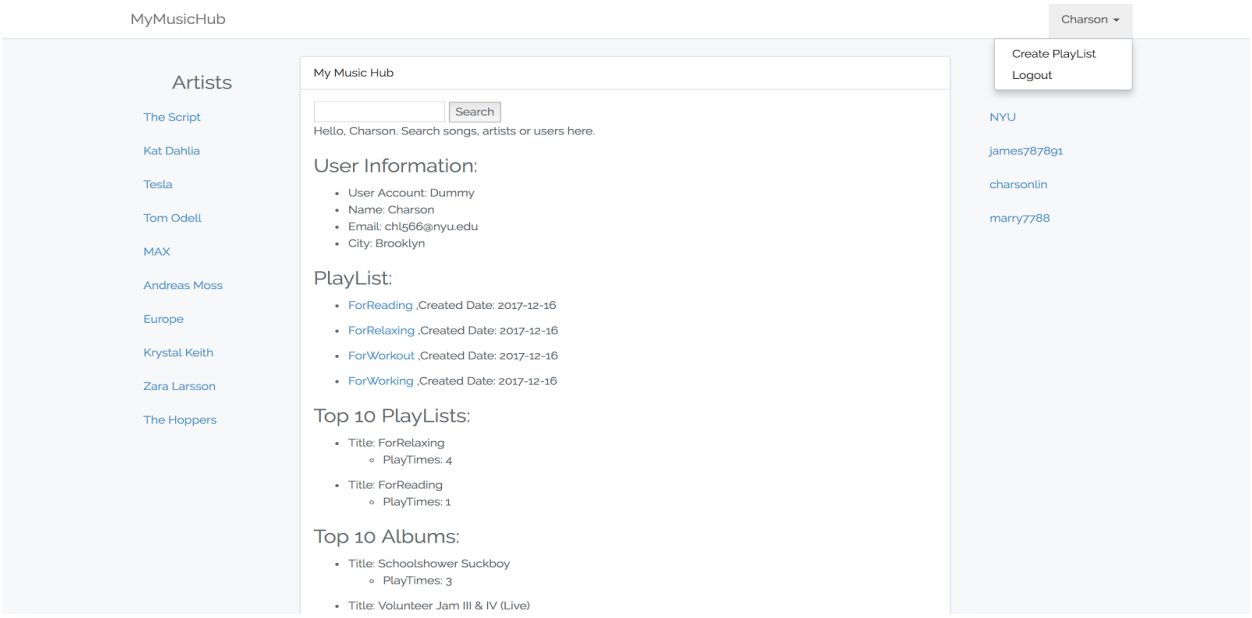
MyMusicHub Charson

MyMusicHub

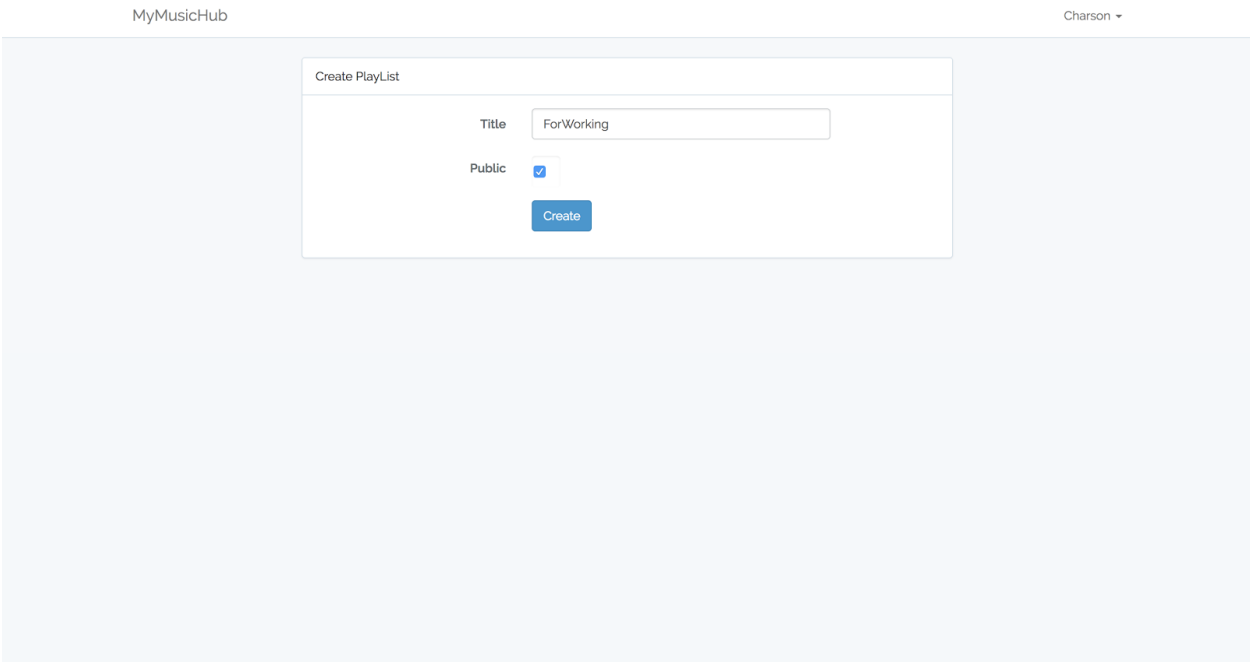
PlayList Title: ForReading

- Tile:Separator (Four Tet RMX)
  - Duration: 07:03
  - By Artist: Radiohead

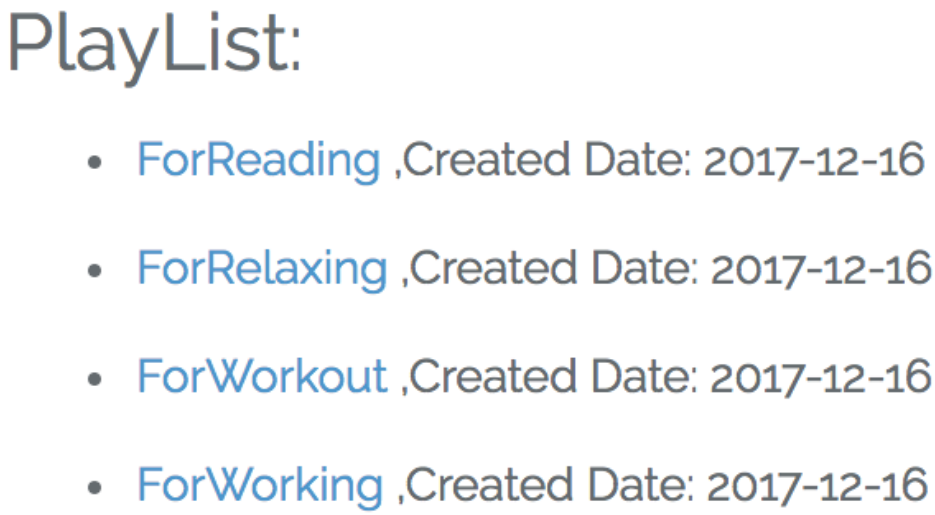
The user could logout by clicking the logout button and jump to the page which allows the user to generate the new playlist.



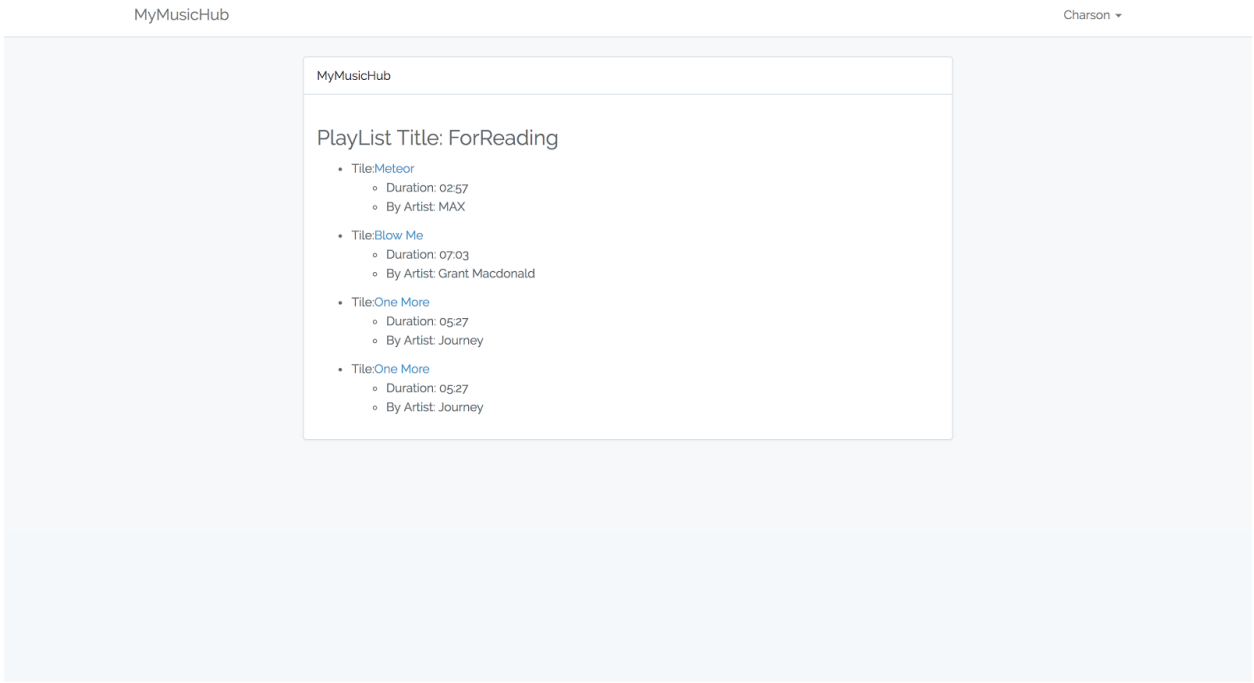
This is the page that the user could generate playlists. The user is allowed to choose if this particular playlist is opened to the public or keeping private. Once the playlist keeps in private, the other users are not able to search or use this particular playlist. Otherwise, the playlist is default opened to the pubic.



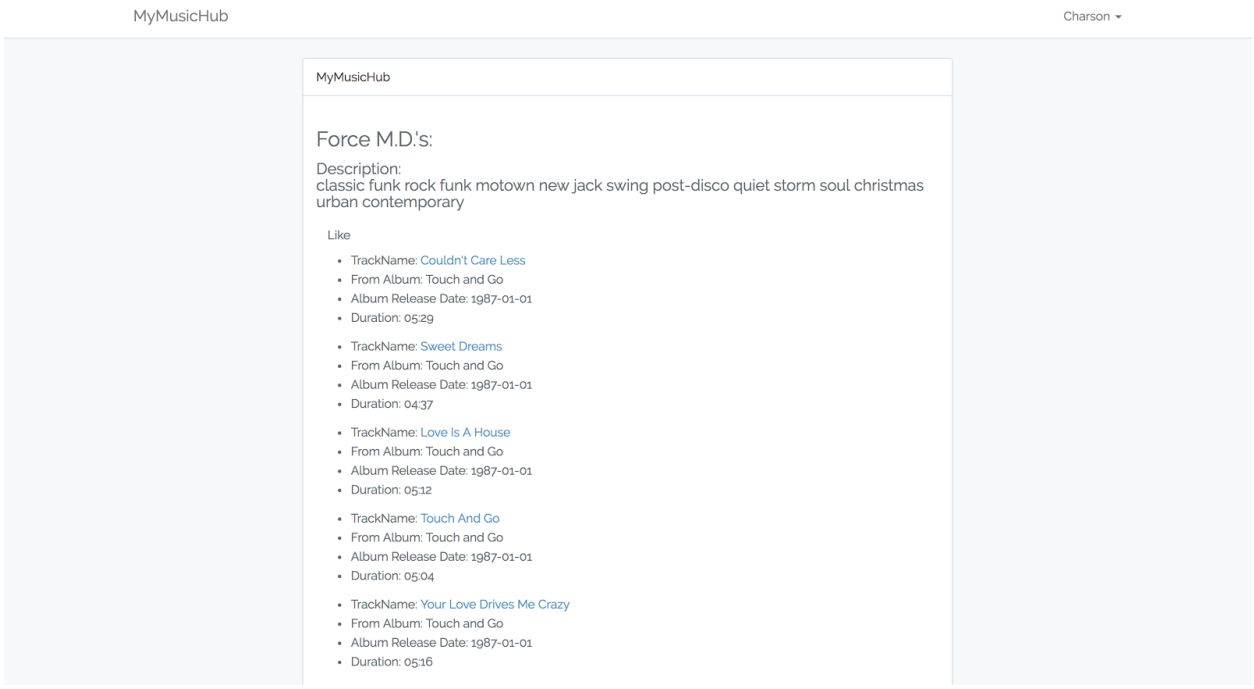
After generating a new playlist, it will reveal on the homepage and the user could add any track as they wish anytime.



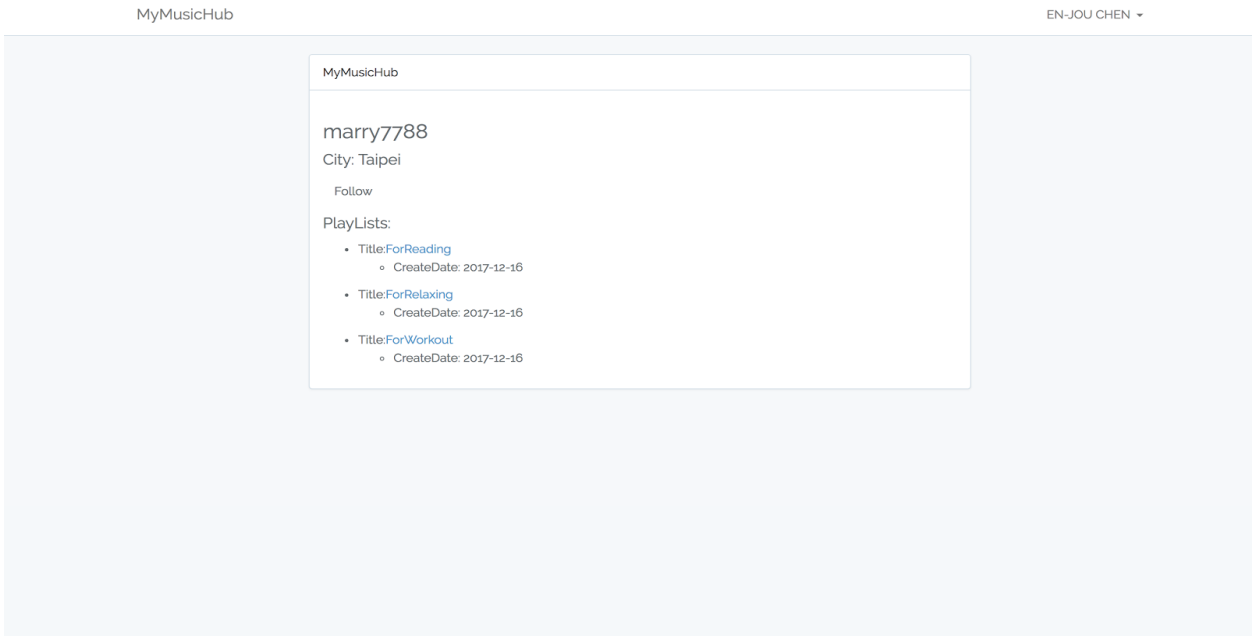
Here is the page after clicking on the playlist list at the home page. The playlist lists a bunch of details about the tracks stored in the particular playlist.



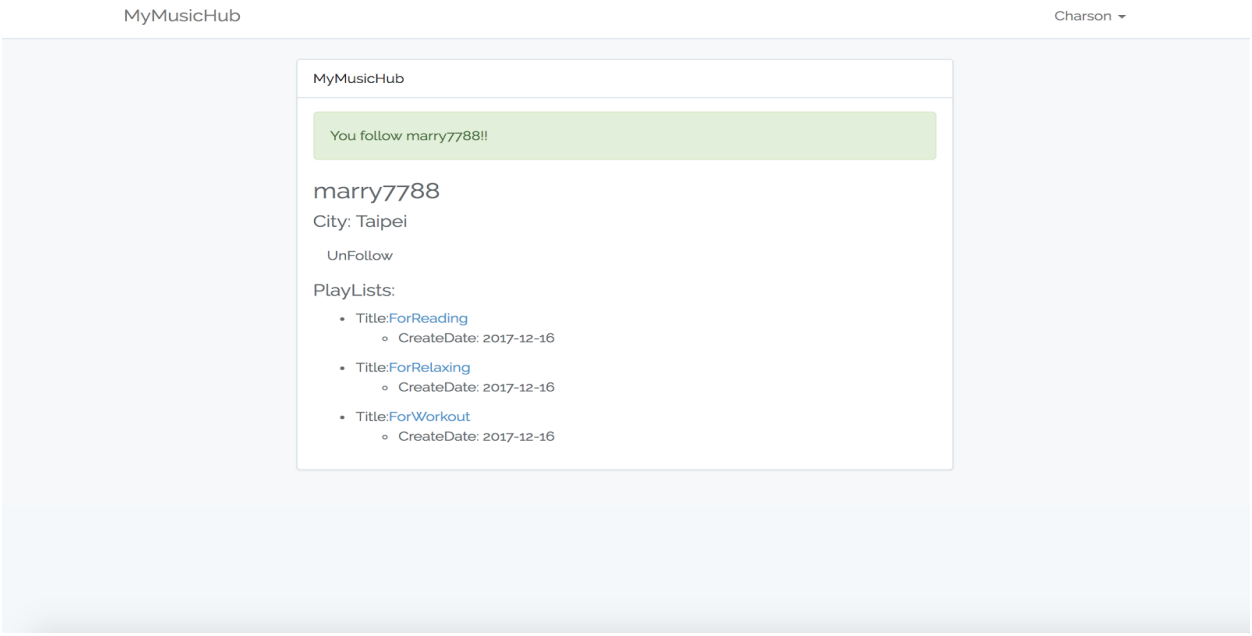
Here is the jumping page after clicking on the left-hand side Artists on the home page. It's basically a artist page similar to the page mentioned above. The user could click the track to play the music and the user is able to like or unlike the artist as he wants. We use this like records as the ranking of listing the artists on the homepage as suggestion to every user.



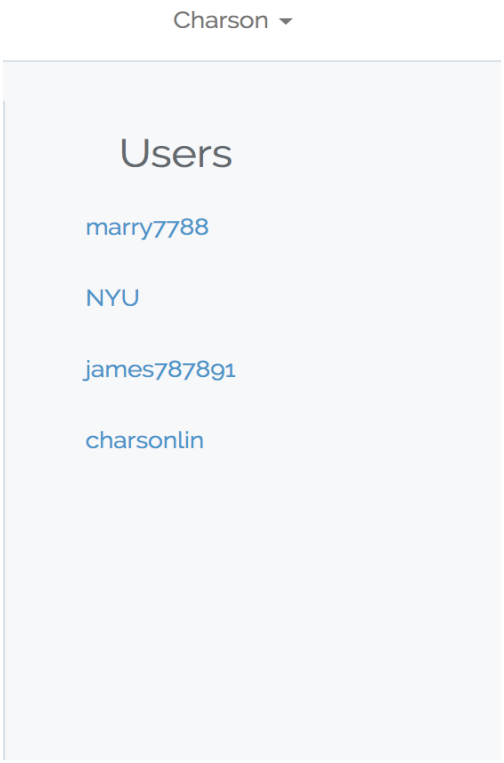
After clicking the users on the home page, the user could get the basic information about the particular user they interested in. Besides, the user is able to click any public playlist generated by the particular user and further into their playlist page to listen the music collected by the particular user.



The user is also allowed to follow the particular user or unfollow the user anytime. Once the user follows the particular user, our system will list those users have been followed by the user first on the right-hand side of the homepage.



As you can see on the home page, the user ‘marry7788’ list on the top of the user list after the user follow ‘marry7788’.



Users should be able to perform all operations via a standard web browser. This should be implemented by writing a program that is called by a web server, connects to your database, then calls appropriate stored procedures that you have defined in the database (or maybe send queries), and finally returns the results as a web page. You can implement the interface in several different ways. You may use frameworks such as PHP, Java, Ruby on Rails, or VB to connect to your backend database. Contact the GAs for technical questions. The main restriction is that the backend should be a relational database using the schema you designed in the first part, with suitable improvements as needed.

Environment setup and Plan:

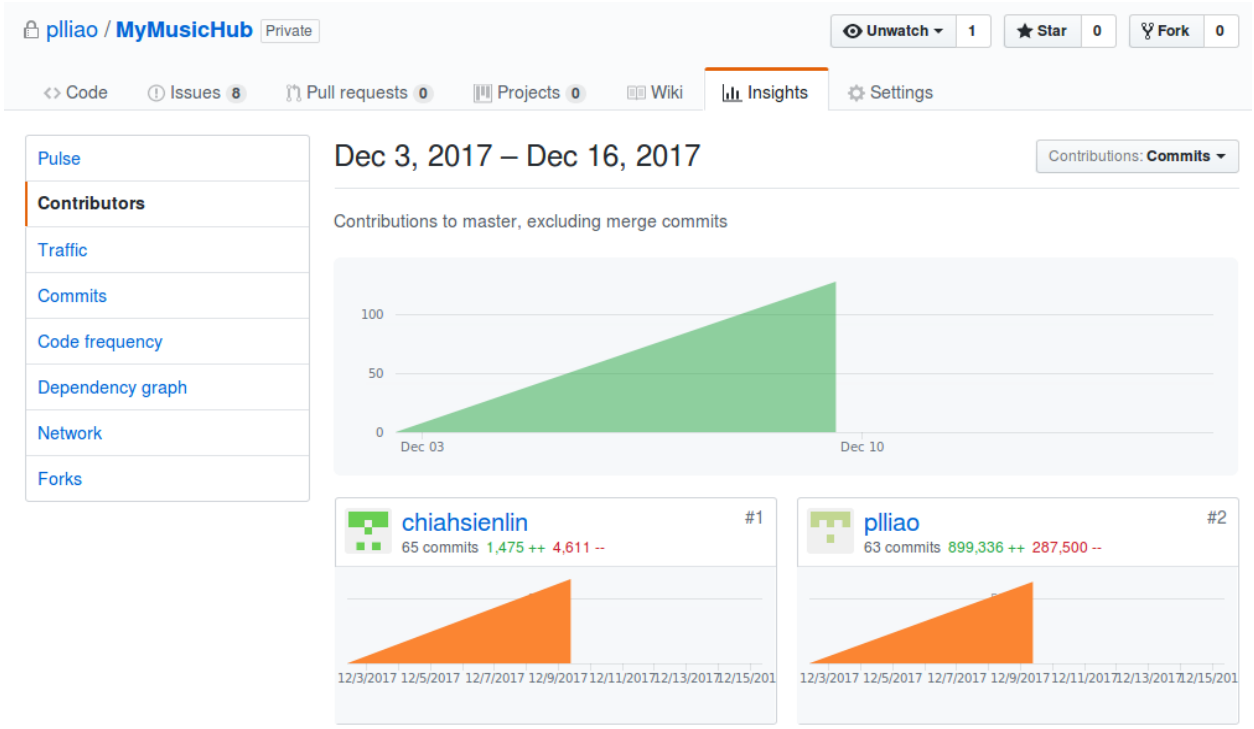
In the project, we use Laravel PHP framework to build the website. The environment of our systems are listed below.

- Pei-Lun Liao
    - PHP 7.0.26
    - Laravel 5.5.25 with Homestead VM
    - MySQL 5.6.33
    - Ubuntu 14.04 LTS
- Chia-Hsian Lin
    - PHP 7.1.7
    - Laravel 5.5.25 with Homestead VM
    - MySQL Ver 14.14 Distrib 5.7.19, for macos10.12
    - macOS High Sierra version 10.13.1

We decide to choose PHP as our primary programming language since we only have experience in building websites in PHP. And we decide to build it with a standard web framework for two reasons. The first reason is to learn how to do web programming in a modern way. Using a framework could help us get familiar with the latest technology in web such as MVC design pattern, user authentication, Url routing, password encryption and reset, and so on. Another reason is that web framework usually has strong support in security to prevent SQL injection, Url Injection and cross-site scripting. Also, we can find resource easily from the community if the framework is popular. And since Laravel is one of the most popular web framework in PHP, we decide to work with it in our project. We also use two PHP open source libraries to help us speed up the development.

- flynsarmy/csv-seeder
- laravelcollective/html

We cooperate on Github and make the project portable and reproducible on different environment and machines. The project is on my private repository, plliao/MyMusicHub, on Github. The contribution for each member could be traced with the git log.



Data processing and Database building:

In this section, we describe how we process data and how we load the data into database. We noticed that there exists encoding issue in the raw data. For example, the character É in the BEYONCÉ [Platinum Edition] album was encoded into ed8a. However, the correct UTF-8 encoding way is c389. We find there are plenty wrong encoding in our dataset. Here are some of the example.

- ÷ in ÷ (Deluxe) album was incorrectly encoded into ed87.
- Ø in quiet is viølent ep album was incorrectly encoded into ed9f

It causes a challenge to load the data into database. Since we try our best to keep the migration process smooth and fast, we decide to insert tuples into database in a batched way. In other words, we insert a thousand tuples into database at once for many times. If there is an insertion error, we will lose plenty of correct tuples. This is because our database will reject the whole batched insertion. Therefore, we have to remove the invalid encoded character first.

Also, we noticed that duplicate tuples exist in the dataset. It will cause primary constraint error in our database. Hence, we sort the data with the id in each file and remove the duplicate line. The command to process the data is the follow one.

iconv -c -t UTF-8 < file.csv | (head -n 1 && tail -n +2 | sort) | uniq -w 22 | sed -e "s/^M//" > processed_file.csv

Iconv will remove the invalid encoding characters in file.csv. And we pass the file with pipe to head and tail process. Head will keep our header in the csv. Tail will put the data into sort process. Therefore, we will have our header with the sorted tuples. Then, we remove duplicate id in the sorted table with uniq command. The parameter -w 22 means only compared with the first 22 characters which is the length of id. Finally, we remove ^M carriage return at the end of line with sed.

Unfortunately, there still exist some challenges in our dataset. The title of track, album or artist may contain the delimiter character, comma, in the csv file. For example, “Hank Williams, Jr” is the name of a artist. Also, "Big City - Live At Brick Breeden Fieldhouse, Bozeman, MT / March 22, 2017" is a track title. The comma in the tuple will cause the insertion error in database due to the foreign key constraint or type constraint. And it will not be the data we want in our machine.

I deal with the comma issue in a trial and error way. First, replace comma with space in some pattern like comma+space, comma in double quotes or comma + Jr. And manually remove illegal comma while the database complains about the input data.

However, there still exist a problem in building the database. Some albumId or ArtistTitle in track.csv are not existed in Album.csv and Artist.csv. Hence, I have to remove the irrelevant tuples first. Otherwise, a foreign key constraint error may occur. I use pandas, a data framework in Python, to solve the issue. I joined three tables on their id key, removed missing tuples and saved the inner joined table as my data to load into database. After that I can load my csv file into database without error.

```
36      data = track.join(
37          artist.set_index('ArtistTitle'),
38          on='TrackArtist',
39          how='inner'
40      ).join(
41          album.set_index('AlbumId'),
42          on='AlbumId',
43          how='inner'
44      )
45
46      data[[
47          'TrackId',
48          'TrackName',
49          'TrackDuration',
50          'TrackArtist',
51          'AlbumId',
52          'ArtistId'
53      ]].dropna().to_csv('tracks3.csv', index=False)
54
```

We use an open sourced project, csv-seeder, to load csv file into database. The advantage to use seeder in building database is that we can migrate or refresh our data easily. The loading flow is below. We will read each row in the file and as we reach the insert_chunk_size number we can insert the data into database.

```
181                 $row = $this->readRow($row, $mapping);
182
183                 // insert only non-empty rows from the csv file
184                 if ( !$row )
185                     continue;
186
187                 $data[$row_count] = $row;
188
189                 // Chunk size reached, insert
190                 if ( ++$row_count == $this->insert_chunk_size )
191                 {
192                     $this->insert($data);
193                     $row_count = 0;
194                     // clear the data array explicitly when it was inserted so
195                     // that nothing is left, otherwise a leftover scenario can
196                     // cause duplicate inserts
197                     $data = array();
198                 }
```

After implemented seeder for albums, tracks and artists, we can simply use the command to migrate the database.

php artisan migrate --seed

Design Enhancement:

- User table
  Since user salt will be automatically generated and stored with the encrypted password, we don't need another column to handle the salt in Laravel.
- Likes, Rating, Follower
  In Laravel, the standard way to handle the timestamp is using two column, created_at and updated_at, to record the tuple history instead of a single timestamp. Hence, we follow the standard way to record our log.
- Album, Track, Artist
  We find that the lengths of artist title, album title and track title are longer than the length we expected. Hence, we update the VARCHAR length from 45 to 200.

- PlayByPlayList, PlayByAlbum
  We notice that storing play history in two tables and tracing the user action are a tedious job. And we find that we didn't store the username in play history. Hence, we can't display the users' history on their dashboard. A better improvement is logging the user action in one table, storing their username and use a column, source, to distinct the play from album or playlist. However, we didn't implement the idea in this time.

Your interface must take appropriate measures to guard against SQL injection and cross-site scripting attacks. To prevent SQL injection you can use stored procedures and prepared statements (if your programming language supports them). If your language does not support prepared statements, your code should check and sanitize inputs from users before concatenating them into query strings. To guard against cross-site scripting, outputs to be returned to user's browsers should be checked/sanitized to avoid scripts. Some languages provide functions, such as PHP's htmlspecialchars, to help with this. You should also define appropriate transactions to make sure that multiple users can use the site at the same time. Make sure to address these two requirements (protection against SQL injections etc., and concurrence) in your submission.

Security and Concurrency:

Avoid CSRF attack:

Laravel automatically generates a CSRF "token" for each active user session managed by the application. This token is used to verify that the authenticated user is the one actually making the requests to the application. This enable our system automatically verify that the token in the request input matches the token stored in the session.
CSRF (cross-site request forgery) tokens are used to ensure that third-parties cannot initiate such a request. This is done by generating a token that must be passed along with the form contents. This token will then be compared with a value additionally saved to the user session. If it matches, the request is deemed valid, otherwise it is deemed invalid. As you can see the {{ csrf_field() }} token bellow.

```
<form class="form-horizontal" method="POST" action="{{ action('UserPageController@follow') }}">
    {{ csrf_field() }}
    <input id="username" type="hidden" name="username"
        value=<?php echo $userinfo->username; ?> />
```

Avoid SQL Injection:

Laravel's Eloquent ORM uses PDO parameter binding to avoid SQL injection. Parameter binding ensures that malicious users can't pass in query data which could modify the query's intent. PDO parameter binding is a PHP function binds a PHP variable to a corresponding named or question mark placeholder in the SQL statement that was used to prepare the statement.

Transaction:

In our project, no concurrence will occurs and we don’t need transaction because we do not have continuous read and write queries. However, Laravel provides simple and convenient function to avoid this kind of problem. As the following code you can see, we can use transaction method on the DB facade to run a set of operations within a database transaction. If an exception is thrown within the transaction Closure, the transaction will automatically be rolled back. If the Closure executes successfully, the transaction will automatically be committed. We don't need to worry about manually rolling back or committing while using the transaction method.

```
DB::transaction(function () {
    DB::table('users')->update(['votes' => 1]);

    DB::table('posts')->delete();
});
```