CSC7084 – Report by Luke Wyers

Website Name: Rockin' Reviews

URL: http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/main/index.php

Video Walkthrough: https://web.microsoftstream.com/video/ad7be05a-1366-4204-9a89-225fd7405064

Table of Contents

Project Structure ..... Page 2
System Requirements ..... Page 2
Libraries, Frameworks, Boilerplates ..... Page 6
Dataset Improvements ..... Page 7
Database Creation ..... Page 7
API Calls ..... Page 7
Bibliography ..... Page 8
Appendices ..... Page 9

Project Structure

The general structure of this project (Appendix fig.1) starts with the root folder named *'rocking-reviews'*. Each part of the site is then divided into main sections. Each section contains an *'index.php'* file which acts as a landing page for that section – for example, the 'album-filter' directory's *'index.php'* contains links to all album filter options such as, filtering by album, artist, etc. On this level of each section are the main webpages. All of these webpages interact with a number of other files which are contained in separate directories. Each section directory also contains a separate stylesheet folder named *'css'* and a file containing JavaScript files names *'js'*. One stylesheet was used per section and any specific styling that was needed was inserted into the html tags. This was only done when JavaScript was involved to show or hide sections of the website such as hiding results from a search query. A single JavaScript file for a section was similarly added in the *'js'* folder. This contained the full script used on all files in that section – each adding them into the end of the body element using a script tag to link them. All sections included a directory named *'includes'* which contained a file dedicated to php functions for that section. This ensured that the files that displayed web content were not filled with lines of php code and allowed for code reuse scenarios. Within *'includes'* there are process files which will perform a specific task, for example, *"album-delete.inc.php"* deletes all *genres* and *subgenres* from an album, then deletes the given album. Having an individual files for a process restricts which information can be sent to that file and makes the code more maintainable. Finally, a file named *"dbconn.php"* is included within the *'includes'* directories. This file provides credentials to connect to the database. Contained in the root directory is 'API' which provides the endpoints for all API calls used throughout the website.

System Requirements

Tier Users

Three tiers of users were created; guest, user, and admin. Sessions were used to differentiate users and the super-global *'$_SESSION'* was used to authenticate users. On sign-up a user is automatically assigned as *'userTypeId=1'* which defines them as a regular user. Guest information was not recorded. Admin users set by changing *"userType" to 0 in the database. Adding a superuser that could create and maintain admin users would be implemented if this project was done again. At each relevant page the session is started. The session will also differentiate the navigation bar that is shown. A guest will not be able to view the profile page and all associated pages to a users profile. This is the same for guests and users being unable to view admin user only pages. This was implemented by checking that the set '$_SESSION['userTypeId']' was correctly set for admin users. By using a conditional statement, if the user did not provide the correct credentials, then they were sent back to the main landing page. This feature was used frequently throughout. In the album description page, features such as adding a score are greyed out with a link to the login and only accessible when logged in as either a user or admin user. The button to add a review is also disabled if guest.*

Sign-up, Login, Log-out

The processing for this was influenced by a video walkthrough on how to implement a login system [1]. The potential user enters their name, new username, new email, and create a password. The username and email both have to be unique values in the database. If non-unique values are inserted for these a flag is shown when trying to sign up. Validation is used for passwords by including a second field to ensure that the user entered the same password twice. The password also includes the constraint of 8 characters minimum and necessitating at least one uppercase and one lowercase letter. The password also needed to have at least 1 number. After signing up, a message is displayed that the sign up was successful, with a button containing a link to the login page. For logging in a user can provide either their email or username. Once they have logged in, they are redirected to the main index page and their username is shown on the navigation bar. A user can sign out at any time from the navigation bar and when they do they are redirected to the main home page.

Albums – View, Filter, and Search

In the main landing page, the user can choose to be redirected to the filter pages for *'artist', 'genre', 'subgenre',* and *'year'.* The main page for filtering albums can be accessed using the link in the navigation bar. The main page shows all albums at once with album artwork. They are displayed in order of their Rolling Stone ranking. If a user clicks on the album artwork, they are redirected to the album's page in which more details are shown. The user can hide all albums at any time. Four links for the filtering options above are presented along with a link to *'search'. 'artist', 'genre',* and *'subgenre'* following the same structure. A button that shows the alphabet can be activated for any of these pages and the page will display any albums that contain the first letter of that filter. For example, choosing 'A' on the 'subgenre' page would return all albums that have 'Acid-Rock' and 'Alternative Rock' as subgenres. The other button for these filter the categories' distinct values. For example, the "All Artists" button will return all of the artists in the database as buttons and the choice will show the selected item filtered accordingly. The 'year' also contains two buttons - *'decades'* and *'years'*. They work similarly as above first showing all albums from a desired decade, then a user can choose year to select a single year. When each album is shown, the album name and the chosen category filter is shown below the album artwork. For example, if 'decade' in 'year' was selected, all relevant albums alongside the name and release date of the album would be shown. The user can also choose to show and hide all results after filtering for each category. Searching for albums can be accessed from the main album filter page, but it can also be accessed at any time using the navigation bar link. Searching is implemented by using queries for the above filter categories. These can then be shown or hidden depending on the results. For example, if the letter 'a' was searched all records containing the letter 'a' would be returned. The option to show/hide gives added agency to the user when searching for an album. The search box is handled by JavaScript to ensure that if a user enters a space, then it is replaced with a *'%'* to help manage API calls effectively. As with the filtering, any album shown contains a link to its own display page with more functionality. The above search function is also utilised when an admin user is editing albums. These are handled by API calls.

Album Scores, Favourites, and Owned

A user can add a score to any album, add if they own it, and add to their favourites. This was implemented using icons. The average score is generated using an SQL query to gather all scores from users. This is then displayed on the albums page using stars. The average is given to one decimal place and a half star is given if the average number not whole. A user can add the score for the album by clicking on the relevant star to produce a score out of five. Then a button confirms this. There is an icon of an unfilled heart for ‘favourite’ and a greyed-out dollar sign for ‘owned’. The user can click on these and there is a button underneath to confirm their updated choice. On the main landing page, these are used to show the most popular user albums. Using SQL queries a ‘most favourited’, ‘highest scored’, ‘most owned’, and ‘most reviewed’ album are displayed. Once a user has ‘favourited’ or ‘owned’ an album they can navigate to their profile page where they can view all albums that are ‘owned’ or ‘favourited’. A show/hide button is used for both categories.

Reviews

All registered users can review an album. All ‘accepted’ reviews are shown underneath the album information. If a user has an accepted review, it will be shown as the first review. The rest of the reviews are ordered by the date they were submitted. A review will contain the review title, review body, reviewer's username, and the date posted. If a user has not reviewed the displayed album or their previous review was declined, a button is displayed to add a new review. An input field for title and a text box for the body is displayed. This is managed by JavaScript and the review can be discarded at any point. The body of the review is limited to 250 characters. Once the review is complete there is a button underneath to send the review to the admin team for review. The review is then shown at the top of the reviews, coloured grey with a label of “pending” until it is accepted or declined. A user can go to their profile and show all pending and accepted reviews. The reviews are shown in full on the user’s profile and a show/hide button is used for accepted and pending. The review is automatically set as pending when it is entered into the database using SQL’s default function. An admin user’s admin profile has a link to show all pending reviews. This will show all pending reviews in full with username and timestamp. The admin user has a choice of either accepting or declining reviews. If declined, the record of the review will be deleted from the database.

User Information Management

A user’s profile page shows all of their personal information given at sign up. The password, however, is not shown and contains asterisks instead. When editing their profile, a user is presented with buttons of all editable information. Once a button is selected it will display the choice’s name and a text box to input the updated information. There is a confirmation button. There are validation checks such as ensuring that a user still adheres to password strength explained above and enters the password twice. A user cannot change their username or email to one that has already been registered. This is the same as the sign-up process validation. Once a field has been updated, the user is redirected to their profile page. All this functionality is included for an amin user’s profile. A user can also delete their profile and this section is at the

end of the page. If a user decides to delete their profile, then an additional confirmation is shown. If the user confirms this, then the account is deleted, and the user will be redirected to the main landing page. Due to the foreign constraints in the database, the user's information must be deleted sequentially. A user is told that all their information including revies and added scores will be removed. The data associated with a user's scores, reviews, favourites, and owned albums must first be deleted. Then the user record can be deleted.

Administration User Accounts

An admin user is provided a separate profile page that can only be reached while signed in with the correct credentials as explained above. There is a separate profile for admin accounts which includes links to a page for editing and deleting user profiles. On this page the admin is presented with all users in a table with all their details. There is an edit and delete at the end of their name within in the table. Admin users are also shown in this table, however, the delete and edit buttons are disabled and have the strikethrough affect to show this. If the admin chooses to delete the user, it will instantly be deleted. The code behind deleting a user is like how a user can delete their own account. The database must first delete any scores, reviews, favourites, and owned albums as these are stored in tables with foreign key restraints. After deleting, the page redirects to main admin page. If an admin user edits an account, they are presented with the same process as above for when a user would edit their own account. All information can be changed, and all validation checks are consistent. After the edit has been made, the admin user is shown the updated user table. A link is given to approve/decline reviews. There is a show/hide all pending reviews button on this page. Each review is shown in full with two different coloured buttons to either approve or delete the review. After the choice by the administrator is made, they remain on the same page and can make several decisions pertaining to reviews without being redirected.

Administration Albums

On the main admin profile page an admin user can edit, add and delete albums. To add or delete an album the above-mentioned search was used to help a user find which album they wished to update. The link for the album redirects to a page providing the user with three choices instead of the regular album display page. 'Edit', 'delete', or 'Go back' are the three options. The album name is presented at the top to ensure the user that they are editing the correct album. If the user chooses 'Delete' then they are presented a confirmation before completion. All associate genres and subgenres must first be deleted before the album is. The same for all user scores, owned, favourites and reviews made for the album as well. This is due to the foreign constraints on the album. After these are deleted all information from the 'albumInfo' table are then deleted. Editing an album is implemented similarly to how a user edits their profile. All album info details are included as buttons and with the use of JavaScript a user is given an input field and a confirmation button for their choice. The *'genre'* and *'subgenre'* edit systems work differently. For each of these the user is shown a table containing all associated genres/subgenres. The table contains a delete button which will remove that field associated with the album. The user then can add a new genre to the album at the

bottom of the table using the ‘Add’ button. This was added as albums can have multiple instances of these fields; therefore, a simple edit field would not suffice. When adding a genre/subgenre record to an album there first needs to be a check if these names already exist in the system. If they do not, then a new record is added in the genre table and then they are associate in their relevant table using *‘genreId’ and ‘ablumId’.* This is an example of how information is maintained in a relational database with many to many relationships. After an edit is made the user is redirected back to the page which allows the user to either edit the album or delete it. The user can also choose to go back to the main admin profile after this. Finally, on the main album admin page a user can add an album. This is presented very simply with input boxes associated with each field in a form. A user can only add in one value per field. The fields include “Album Name”, “Album Artist”, “Rolling Stones Ranking”, “Album Year”, “Genre”, and “Subgenre”. The names of the artist, genre and subgenre fields were all checked in the database before adding. This was so that these fields could remain unique. If they were already in the database, then the id associated with their name was retrieved. If not, then a new record was created, and that id was used when associating them to the album.

Security Concerns

Calls to the database used prepared statements to ensure decreased chances of SQL injections. Information input fields were processed with PHP. Any information relating to a user was passed to the database using the *‘POST’* method to ensure it was hidden from view in the URL. All passwords were hashed when entered into the database to ensure that any user information is secured. Only certain users are able to access certain pages throughout the set using the *‘SESSION’* function in PHP. This ensures that a guest user cannot access the admin pages without valid credentials.

Libraries, Frameworks, Boilerplates

Two external websites were used for styling purposes and the use of these was minimal. Picnic CSS[2] was used for the styling framework. This was used for the navigation bar and a number of elements displayed as cards such as the reviews container. The present buttons were also used, taking advantage of the ‘pseudo button’ extensively. However, the Picnic grid system was not used. Instead, flex box was used. Flex box allowed for greater customisation of how the website looked. It also meant that elements could be styled to specific needs for individual pages. Relying on the picnic grid system would result in a cookie cutter looking website. The Font Awesome [3] library was used for a handful of icons such as the stars, dollar sign, heart, and search bar icon. The site was created with other music review websites in mind. The background colour was consistently kept a light blue throughout as attention should be given to the album artwork. The design is purposefully minimalistic. Vanilla JavaScript was used throughout – there was no need to add either more frameworks as these would be superfluous. The use of jQuery would have been unnecessary for the amount of JavaScript used on the site. All information gathered for completion of HTML, CSS and JavaScript come from *The Odin Project [4]* alongside various websites added in the bibliography [5], [6].

Dataset Improvements

The main improvement was to add album artwork for all albums. This was gathered using Spotify's API to search for an album and gather the album artwork. A tutorial on how to use Spotify's API was used as an initial basis to understand how it worked [7]. Then a program was created to search artworks for all albums using the album name and artist name as search queries. The database was expanded by adding users, scores, reviews, and whether a user owned or favourited an album. The user reviews were populated with a timestamp so that on the main page a *'trending'* section of all recent albums reviewed was displayed. The addition of these tables in the database gave added information to the albums by showing community scores and the highest owned, scored, and favourited as described above. All user information was stored in the database so that the above requirements could be performed. A relational database was created to ensure proper data management.

Database Creation

An original dataset containing 500 albums was the basis of the database. To process this information, several processes were used to clean the data and insert it into the database. Firstly, the file was split into four composite files to make data processing easier. The first contained all album information excluding genres. The file was read using PHP and an array was created off all artists. The '*array_unique()'* was then used to ensure no artists were duplicated. A table was created using SQL and then insert statements were generated using a loop to echo an insert statement containing each unique name. These statements were then run on MySQL. The *'albumInfo'* table was then created the same way; however, a function was created to retrieve the artist Id from the database. This was to ensure normalisation in the first normal form could take place. The insert statements were generated and inserted the same way as artist info was. Both genre and subgenre were put into one file and a function was used to populate an array with all these names. 'A*rray_unique()'* was used again to ensure there were no duplicated entries for genre names. The genre table included all genre names as a subgenre may be used as a genre and vice-versa. The same process for inserting artists into the database was conducted and all genres were added. Then each composite file for genre and subgenre was used to generate the *'albumGenre'*, and *'albumSubGenre'* tables. Using normalisation, these tables were created as one album could have many different genres and subgenres. A one-to-many table was created for both with foreign keys constraints added on *"albumId".* The remaining tables were created manually in SQL and the details can be seen in the appendix [fig 2, 3].

API Calls

There were several API calls created in the 'Api' folder. These dealt with getting information for the search results. APIs were to be further implemented, however due to time constraints only a number were implemented. A full list is shown in the appendices. The search api calls were created for each category - 'artist' , 'album', 'year', 'genre', and 'subgenre'. The search queries contained the query term *'q=*' and search type *'type='. The search term was also processed by JavaScript to replace any spaces with "%" to ensure there were no errors[fig4].*

Bibliography

[1] - https://www.youtube.com/watch?v=gCo6JqGMi30&ab_channel=DaniKrossing

[2] https://picnicss.com/

[3] https://fontawesome.com/

[4] https://www.theodinproject.com/

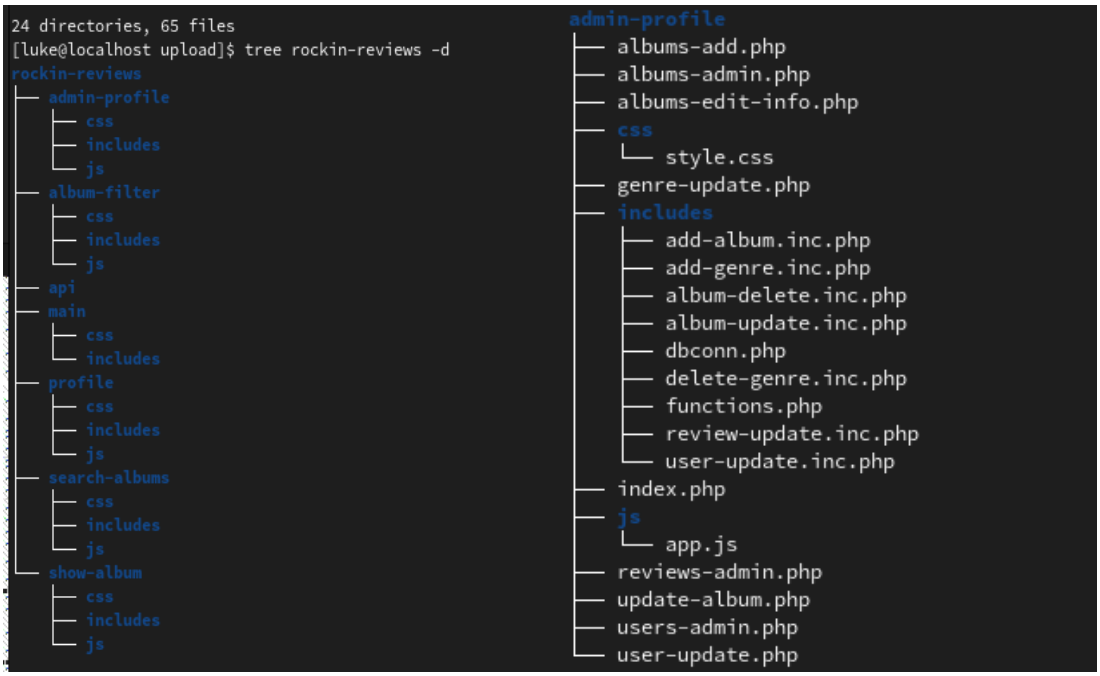
[5] https://www.w3schools.com/

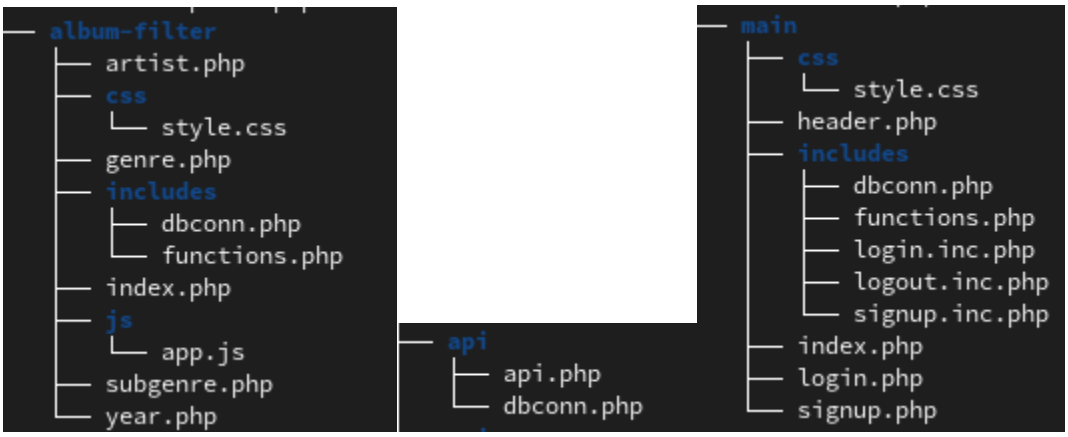
[6] https://www.youtube.com/c/TraversyMedia

[7] https://www.youtube.com/watch?v=1vR3m0HupGI&ab_channel=MakerAtPlayCoding

Appendices

Fig 1 – general Structure





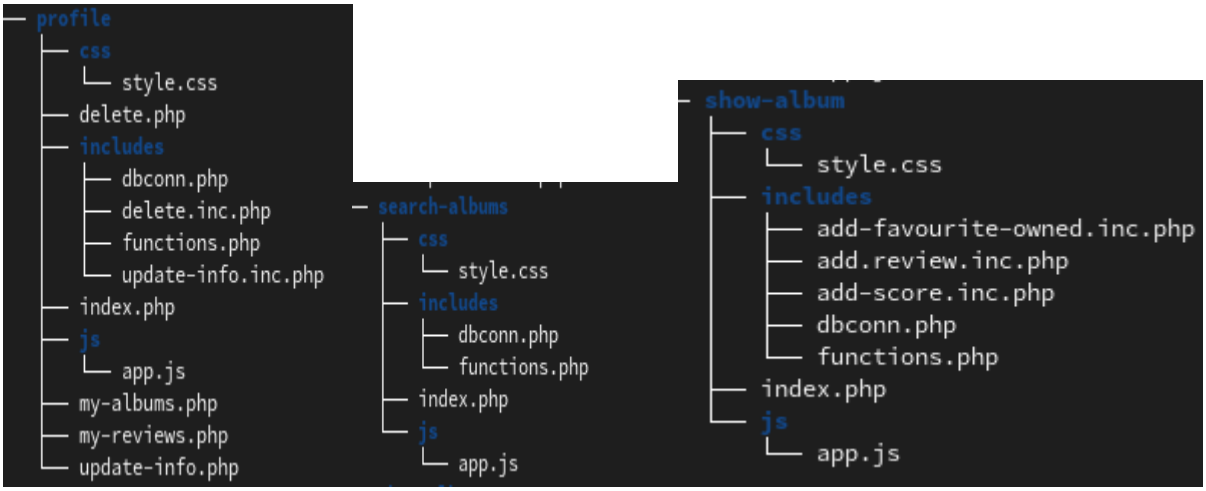


Fig 2 – Database structure

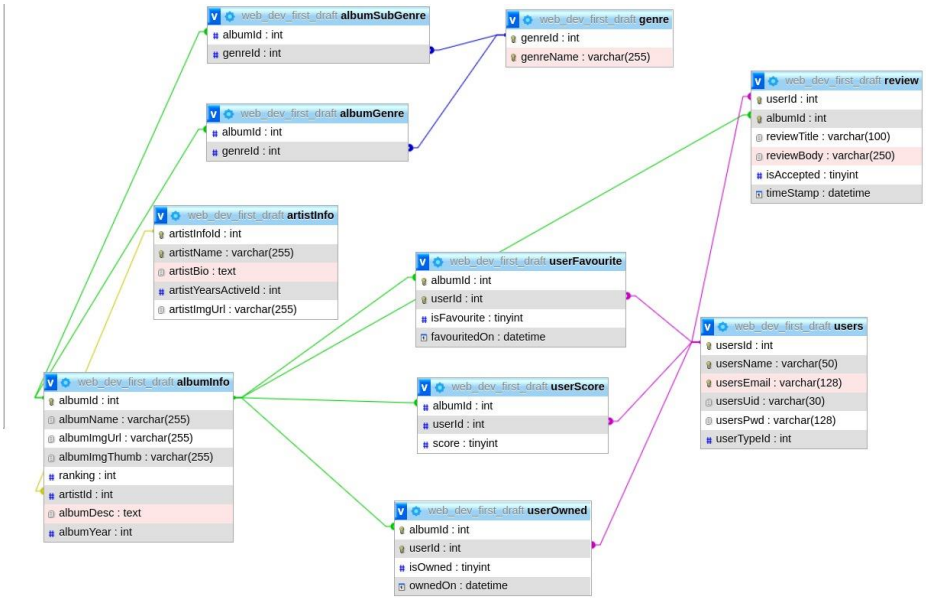


Fig 3 – Database foreign key constraints

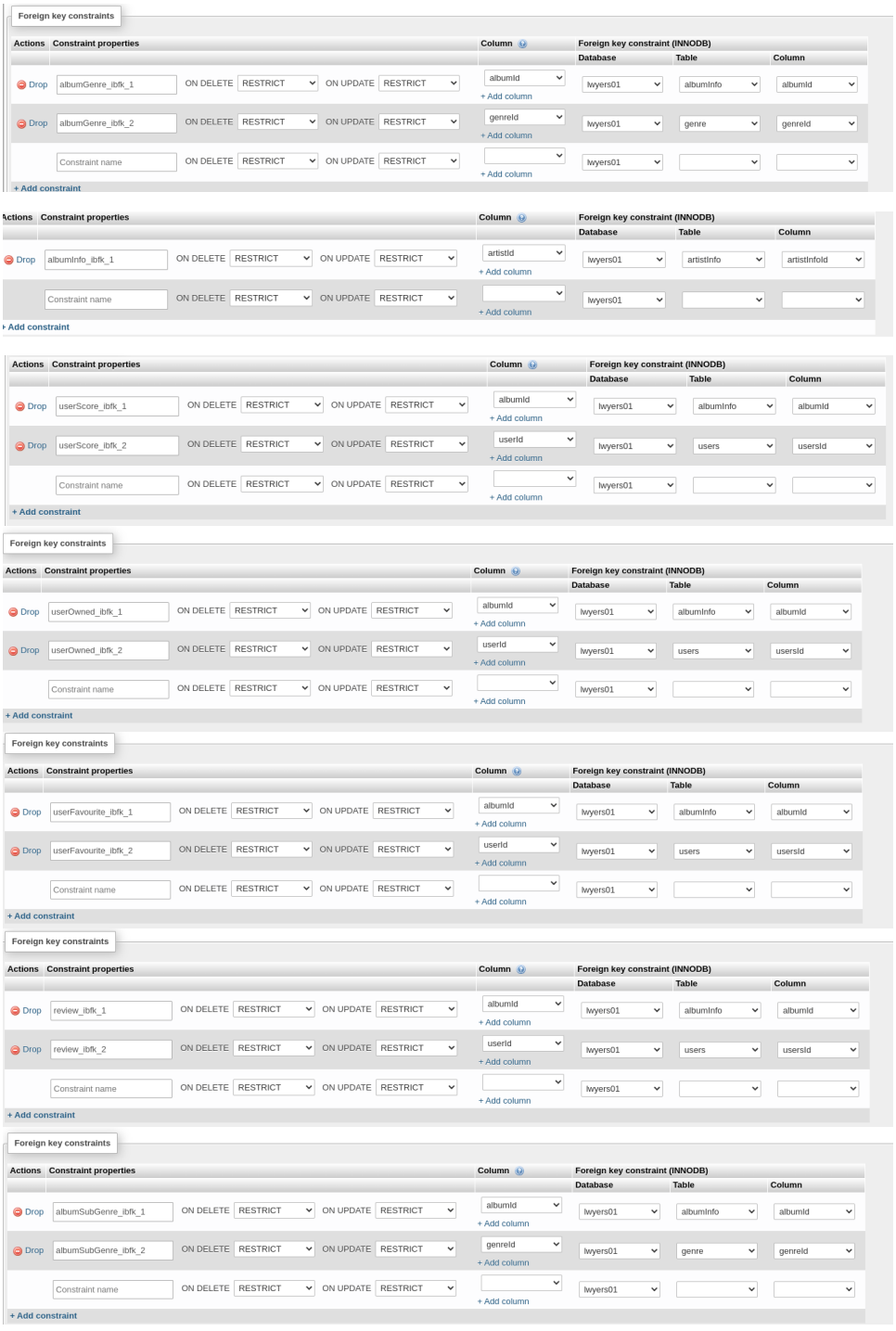


Fig 4 – API endpoints

Get Album Info API

Key: albumId

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?showAlbum=

Get Genre Info API

Key: albumId

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?showGenre=

Get Subgenre Info API

Key: albumId

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?showSubGenre=

Get community rating

Key: albumId

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?commRating=

Get artist info

Key= q=query string, type=artist(ar)

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?q=&type=ar

Get album info

Key= q=query string, type=album(al)

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?q=&type=al

Get genre info

Key= q=query string, type=genre(g)

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?q=&type=g

Get subgenre info

Key= q=query string, type=subgenre(s)

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?q=&type=s

Get Year info

Key= q=query string, type=year(y)

http://lwyers01.webhosting6.eeecs.qub.ac.uk/rockin-reviews/api/api.php?q={$searchYear}&type=y