CSCI 466 Group Project (Spring 2023)

Karaoke Management System

Introduction

This is the group project for 466, in which you will design and develop and application using the tools covered in the class. You should work together in the groups which you chose, or in the groups which will be assigned to you if you did not elect to choose your own. There will be no individual submissions, just the one for a group.

Application

The application you will be designing and implementing will be a web-based, database driven tool to facilitate the running of karaoke events at a bar or other venue. It must allow users to sign up to sing songs from the list of available songs, and also provide the DJ information needed to call up the next singer.

Searching for songs

The potential singer should be able to search through a list of songs by either the artist name, the title of the song, or the name of one of the contributors. A contributor is someone who has contributed to the creation of the song in some significant way, such as the author, the singer, the guitarist, the drummer, etc. As an example, this should allow a user to find all of the songs that Paul McCartney has made contributions to, whether it was as a member of The Beatles or as a member of Wings.

There needs to be information on what contribution each of these contributors has made for each song. As an example, David Bowie would have contributed as a writer, a singer, and likely a guitarist in most of his songs. There are, however, songs that he did not perform, but only wrote, like "All the Young Dudes", which he wrote, but which was initially performed by a band called "Mott the Hoople".

Each song will have at least one karaoke file, or it wouldn't be present on the list (because the karaoke presentation software wouldn't have anything to play), but each song may possibly have many karaoke files, one for each of several possible versions. Each of these files will have a unique identifier. Since it may be important to the karaoke singer which version they are to perform, the version should also be a part of the information used when a user signs up in one of the queues to sing a song.

As an example of multiple versions of a file, think of a duet. There may be a version that is arranged so that both partners get to sing, as well as a version for each of the parts, with the other singer's part dubbed in (in case a singer wants to sing one part and doesn't have a partner).

Signing up to sing

Upon choosing a version of a song, the user should be able to enter their request to sing a their chosen song into one of two queues that are stored separately.

The first queue is a free for all, first come, first served queue that can be entered without charge.

The other queue is an accelerated, priority queue, where the user can pay money to potentially have his song played earlier.

Your application needs to allow people to sign up in the queues for singing. The DJ will decide how to actually choose which song plays next at any given time, but the contents of each of the queues should be shown to him separately.

DJ interface

You are also tasked with designing the DJ interface. This interface should show both queues separately (DO NOT MERGE THEM), but on the page, including relevant information about both the user that is singed up to sing and the version of the song they signed up in the queue to sing. This information should include the user who requested the song, its title, the name of the band that performed it, and the special karaoke file ID associated with the version of the song that was selected.

Web Interface Requirements

The interface for a user signing up to sing must be separate from the DJ interface.

When searching for a songs, it is possible that there may be many rows of results returned. When this occurs, the user should be able to click on a table's column heading to sort the results based on the data value represented by that column. The first click will sort the table in ascending order based on that column, then the next will sort in descending order, then back to ascending again, etc. This problem can be solved with PHP, but there are also other ways of handling it if you'd like to look into other, more dynamic options.

The free queue should be sorted based on the time the user signed up; first in, first out. The accelerated queue should be able to be sorted in either time order or by the amount paid. Make sure to implement some mechanism to switch the sort order.

Allowed tools

Obviously, any of the tools that have been covered in class are acceptable. This mostly means using our MariaDB server (an absolute requirement), and PHP/HTML forms/PDO.

Each semester, I have a lot of people asking me if they can use other tools, such as JavaScript/CSS/etc. The answer I give is that I do not have a problem with the use of such tools, as long as they're not a shortcut that allows you to skip the work of actually doing the project.

This means that any of the code implementing requirements detailed in this write-up must be designed and written by your group. Obviously this would not be the case if you just downloaded and included a library that already did the entire project.

However, outside of the main functionality, it is acceptable to use libraries. Things like Bootstrap, React.js, etc. are acceptable, as long as they're supplementing your work, and not replacing it.

As an example, if your group wants to implement more dynamic functionality with JavaScript and the DOM, any code that interfaces with the database must be written by members of the group.

You can use CSS and JQuery all you want to change the appearance of the webpage, but since these tools were not discussed in class, they will obviously not be requirements.

Some notes

- This is a group project, and you need to be able to coordinate with the others in your group. This is the reason that NIU provides Teams, so I would encourage you to coordinate through it, though this is not a requirement. I know that some of you prefer to use Discord/Telegram/etc. and this is not a problem, but remember that each group is responsible for implementing their own application: code should not be shared between groups, and any discussions should be held in a setting not viewable by people outside of the group.
- This project is designed to give everyone a chance to apply their knowledge, and to learn where the gaps in that knowledge may be. To that end, please make sure that all of your group members are involved in each of the steps. It might be easy to carry/be carried, but it is better for everyone if it's actually a team effort.
- I would recommend setting up some sort of version control repository (`git` is one that is commonly used) to coordinate the coding between members of your group. If you move the database login information into a separate file that is included, you can minimize the amount of code that needs to change between users as they independently test/develop. Students in the past have set up GitHub repos for their group to great effect. Make sure, however, that only your group members have access to the repo.

What to turn in?

Submit, via Blackboard (one submission per group), the following:

1. The ER diagram (in PDF format) that you designed for the database that is used for the application. This should be your first step, and the other steps should be based upon this ER diagram. It would be best to draw this with some sort of software, but if no one in your group can make that happen, then hand-drawn and scanned diagrams will be accepted, but legibility will be critical and points will be lost for anything a grader cannot read. All entities and relationships must be drawn, along with any identifiers or intersection data. Other attributes do not need to be drawn, but must appear in…
2. A description of all of the entities, relationships, and attributes that are a part of your ER diagram. This will obviously include their names, their purpose, and any additional data that is important to know. This can be a part of your ER diagram PDF, or a separate PDF file, but it must be present and easy to find.
3. In a PDF, include the relational schema of the database, converted from the ER diagram. Include information on which attributes are primary/foreign keys, and make sure to identify what the home relation is for the foreign keys used. This can be in the same PDF, or a separate one, but it must be present and in an obvious location. It should be based on the ER diagram from before. We will check to make sure it matches, so make sure that any changes made are applied to the ER diagram and the schema here.
4. An SQL script, suitable to run with MariaDB, containing the DML code to create the database designed and detailed in the previous portions.
5. Another SQL script, suitable to run with MariaDB, that inserts the sample data needed to make your application run properly. This should include at least 30 songs (of your choice), 10 users, and 5 of them signed up to sing in each of the two queues.
6. A tarball or zip file containing the PHP code and any additional web-facing files that implement the application.
7. A web link to your group's application running on the student web server. This can be served from any of the group members' `public_html` on `turing`/`hopper`, but it does need to be working there to facilitate grading. Make sure not to delete the implementation until after grading has been completed.

DO NOT submit Word documents. It is acceptable to write them in Word, but make sure to print/export them to PDF before submission.