Project 3 Documentation

Firoozeh kaveh

How to Start:

The main function in *LibraryServer* class should be executed to run the server. Then by going to local host "/login" page, you can start interacting with the web application.

User Experience:

If the user is already registered, they can use their credentials to log in. If the provided user name and password is invalid, a message will indicate that and asks the user to try again. If the user wants to register for a new account, they can click on the "signup" button, and they are directed to the "/signup" page to register for an account. If the typed in user name already exists in the database, an error message is shown, and they are prompted to log in using the existing account.

After log in, the user observes the "/homePage", in which three options are available. The first option is "Show All Songs". If the user clicks on that, a list of songs saved in our database is displayed in "/showAllSongs". This page is basically a table that lists all the songs with their album and artist name, plus a button to fetch the description of the song, and a button to add the song to the playlist of the current user. By clicking on the "Show Description", we call the theaudiodb.com to get the description for the specific song and show it in a new page. The reason for this design choice is that fetching these descriptions is slow and we want to give the user the option to fetch them only for a selected song. The other button called "Add" will add the selected song to the playlist of the user. After clicking, we show the "/playlist" page, in which only the songs that the user has added to the their playlist is shown.

The other option on homepage is "/search" in which the user can run a search of albums or artists in our database. By typing in the provided textbox, and clicking on the "search album" or
"search artist" button, the user is directed to the "/searchResult" page in which the results of the search is shown. If the album or the artist is not found, an error message is displayed.

The final option on the homepage is a customized "/playlist" which shows the songs that the user has added to their playlist. The user has the option to click on the "Delete" button to remove the song from the playlist.

Implementation Details:

In terms of the implementation details, let's start from the beginning. The signed up user information is stored in a table called users, which holds the user id, user name and password. It is worth noting that the passwords are stored as hashes in the users table, so that it is more secure. Whenever a user wants to log in, we look up that username in our table and match the hashed version of their typed password to see if

they match. If they do, we proceed with the "/homePage". Assuming the login is successful, we write a cookie with a certain maximum age that keeps the user logged in while the cookie is valid. If the cookie expires, we redirect the user to the login page.

In order to hold the discography information, we utilize 3 tables:

- Songs: holds the song id, song name, album id, and artist id.
- Albums: holds album id, album name and artist id.
- Artists: holds the artist id and artist name

The playlists table is special because it contains two main columns of user id and song id. Every record in that table means user *i* has added song *j* to their playlist. This way we can surface personalized playlists for our users leveraging the shared databases for songs info.

When serving the user with different parts of the application, we query these tables possibly joining them based on their primary keys to extract relevant information. The job of handling the input/output communications is given to the class called *IOManager*. It makes a connection to the database and facilitates extracting or updating the tables for us. It also handles calling the REST API for fetching song descriptions.

Sequence Diagram: