COMP 3612
Assignment #2: Single-Page App

Version: 1.1 Oct 30, 2023, changes in yellow
Due Monday November 20, 2023 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to work with JavaScript. The application you will be creating is a viewer of song data from 2016-2019 (sorry about living in the past but this is the data I was able to find easily). You can work in pairs or by yourself. Don't delay finding a partner; if you cannot find a partner to work with, **don't ask me to be in a group of 3**, you will have to work alone. The assignment is very doable by one person, but because the class is so large, this is a way to reduce my marking load.

Beginning

It is your responsibility to read all the assignment instructions thoroughly and carefully. If you are unclear about something, ask me. But before you do, read the material in question again!

Starting files can be found (eventually) at:

```
https://github.com/mru-comp3612-archive/f2023-assign2.git
```

Grading

The grade for this assignment will be broken down as follows:

Visual Design	15%
Programming Design	10%
Functionality (follows requirements)	75%

Requirements

This assignment consists of a single HTML page (hence single-page application or SPA) with the following functionality:

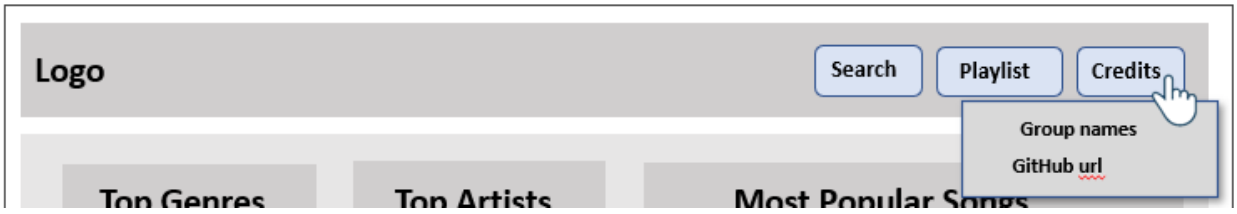
1. Your assignment should have just a single HTML page which **must be** named `index.html`.
2. The assignment should have four main views: **Search/Browse Songs, Song Details, Playlist,** and **Home**. When the program first starts, display the **Home** view.
3. There are two sources of data. The first are JSON files containing a list of genres and a list of artists/groups. Exercise 8.14 in Lab 8 provides instructions for converting a JSON file into a string and then using `JSON.parse()` to convert it into a JavaScript object.

   The other source of data is the song API that provides a list of about 400 songs. Each song includes Spotify analysis information. You will also be provided a JSON file containing a small subset of songs so you can test without the API (though your submitted assignment must consume the data from the API). The API can be found (eventually) at:

   ```
   https://www.randyconnolly.com/funwebdev/3rd/api/music/songs-nested.php
   ```

4. **Make sure you are only requesting/fetching the song data once!** To improve the performance of your assignment (and reduce the number of requests on my server), you must store the song data in `localstorage` after you fetch it from the API (see Exercise 10.11 in Lab 10). Notice that you have to use `JSON.stringify` to create a JSON string version of the array and then saving that in `localstorage`; similarly, after you retrieve it from `localstorage`, you will need to uses `JSON.parse()` to turn it into an array. Your page should thus check if this play data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in Chrome via the Application tab in DevTools). Please do this task early on so that you reduce the load on my server! **Failure to implement this functionality will result in a very large loss of marks so don't neglect it.**
5. You must write your own JavaScript. That is, no jQuery, no Bootstrap, no other third-party JavaScript libraries other than the one used for your charts. You have all the knowledge you need from the lectures and the three JavaScript labs to complete this assignment. If you do find yourself, however, making use of some small snippet of code you found online (and I do mean small), then you must provide references (i.e., specify the URL where you found it) via comments within your .js file. Failure to properly credit other people's work in your JavaScript will result in a zero grade. Using antediluvian JavaScript found online will result in lower marks. There is no need to credit code you found in the labs or lectures.
6. You can use a third-party CSS library. Be careful though that this CSS library is not using/requiring its own JavaScript library as well. Many popular CSS libraries include their own JavaScript libraries to do fancy things with select lists, modals, etc. **You are not allowed to include these** so be sure you don't have any extra <script> tags!
7. **Header**. The header should appear in every view and contain a logo/title, and links for Search, Playlist, and Credits. The screen captures below show these three links as buttons, but you could style them as text links or icon/images instead. Clicking the logo should display the **Home view**; clicking on search should display the **Search/Browse Songs view**; click on Playlist should display the **Playlist view**.

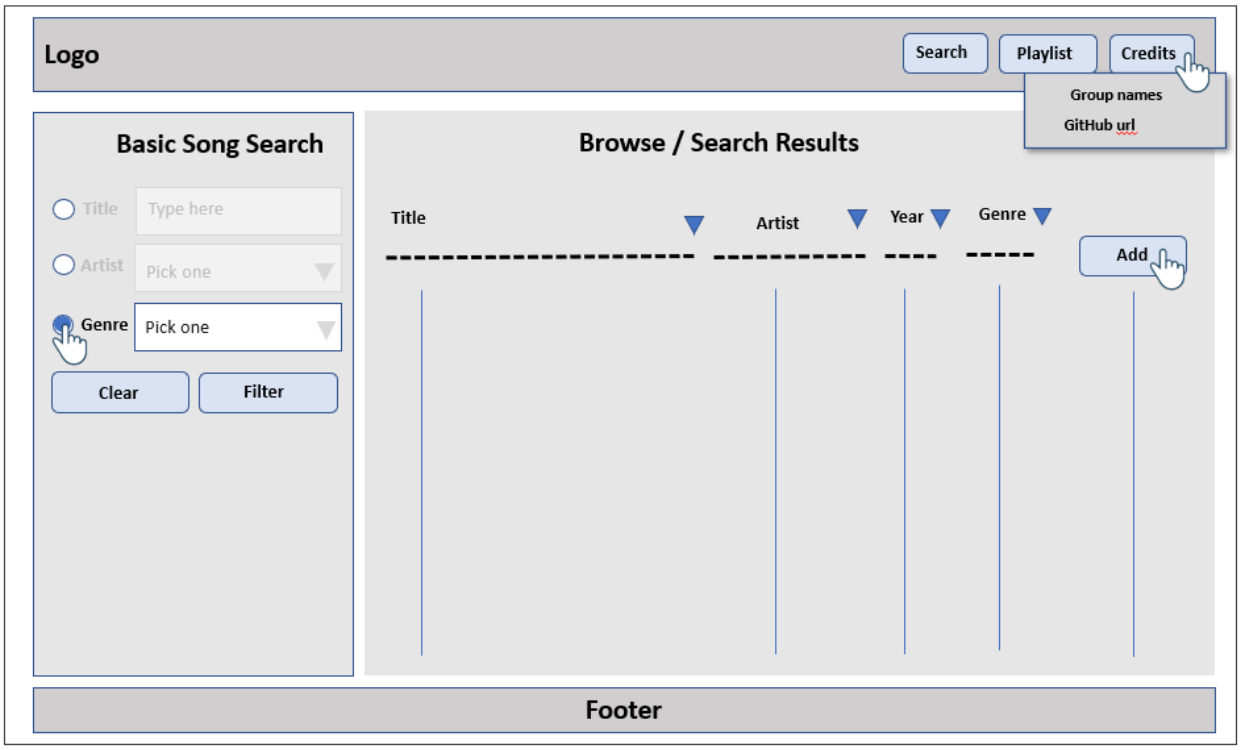
   If the user moves the **mouses over** the credits button/link, then display a pop-up panel that displays the group member names as well as a link to the assignment's github page. It should disappear after 5 seconds (use `setTimeout`). The Playlist button should switch the view to playlist view.



8. **Search/Browse Songs view**. You will implement a page with similar functionality as that described below. The provided sketch below illustrates the functionality. You are welcome to modify the design as you see fit. This view doesn't have to say "Browse/Search Results"; it, and the other bold labels on these pages, are there to explain what functionality must be on the page, not necessarily the actual labels that must appear on your page.

   For each song in the list, display the song title, artist name, year, genre name, and popularity number. Style the song title so it looks like a hyperlink or style the table row and the cursor so each song in the table is clickable. When the user clicks on a song or song title, the view should change to **Single Song View**.

   Initially display the songs sorted by their title. If the user clicks on the column headings, sort the list on the clicked field. The list should refresh whenever the user changes the sort option. Provide some visual cue that a sort change has occurred, such as an icon or styling changing in the header. The list must be an unordered list; ideally, make the list scrollable by setting the `overflow-y` property of the container to `scroll`.



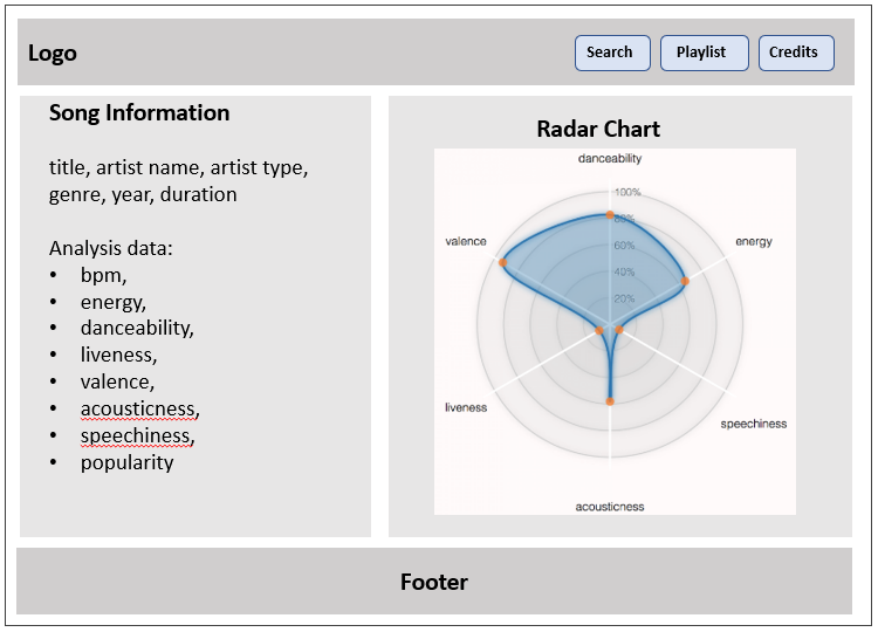
   For the song title, display only the first 25 characters of the title. If the title is more than 25 characters, make the 25th character an ellipse (…) character (use the HTML character entity …). If the user clicks on the ellipse, display the full title in a temporary popup (akin to a tooltip) that disappears after 5 seconds.

   Provide an icon/link/button that allows the user to add that song to the playlist. When the user adds a song to the playlist, display a pop-up panel that tells the user the song was added to the playlist, but hide it after a few seconds. In web design parlance, these transitory popup messages are often called snackbars or toasts. Using a CSS transitions for the snackbar or toast will make it look quite professional. The best way to implement a panel is to define it as a `<div>` in the markup and then programmatically hide/show it using the `display` property.

   The page also needs to provide a way to filter the songs. The user should be able to filter the songs by title (does the word appear anywhere in the title?), artist, or genre. These options should be mutually exclusive and be enabled/disabled based on the current state of the radio

buttons. The radio button should initially default to title. The clear button should remove any filters. You will find the array `filter()` function will be your friend here. Be sure to disable filters based on the current state of the radio buttons.

9. **Single Song view.** I would expect you to group and present this data in an intelligent way. For instance, no user would like to see the duration listed as it is in the database (number of seconds); instead they would expect to see number of minutes and seconds. Look at how information is presented in commercial web dashboard sites for styling inspiration. Don't be afraid of boxes, font contrast, color contrast, progress bars, etc. I would recommend for sure making use of icons to help user identify the analytic values.



For each song, there is a variety of Spotify music analysis metrics, which are:

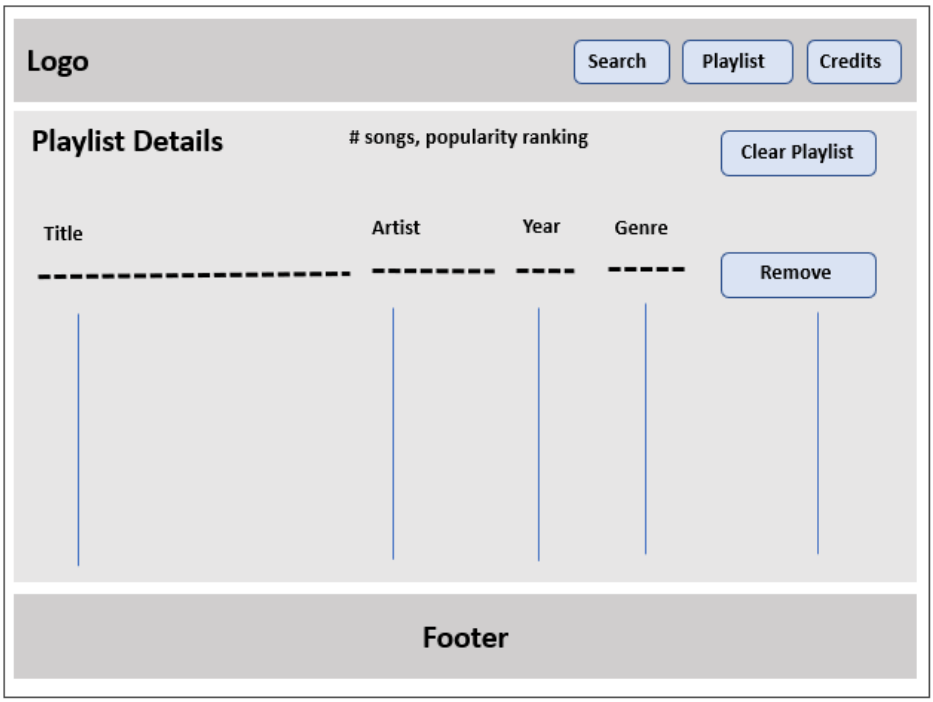
- bpm: The overall estimated tempo of a track in beats per minute (BPM).
- energy: Represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Value between 1 and 100.
- danceability: Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. Value between 1 and 100.
- valence: Describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry). Value between 1 and 100.
- liveness: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. Value between 1 and 100.
- acousticness: A confidence measure of whether the track is acoustic. Value between 1 and 100.
- speechiness: This detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 100 the attribute value. Value between 1 and 100.

- loudness: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Not used in this assignment.
- duration: The duration of the track in seconds.
- popularity: The higher the number, the more popular the track.

Display danceability, energy, valence, speechiness, loudness, and liveness in a radar chart. You can decide the order of the different axis values, the colors, the style, etc. I would recommend using the chart.js library (https://www.chartjs.org/), though you are welcome to use something else.

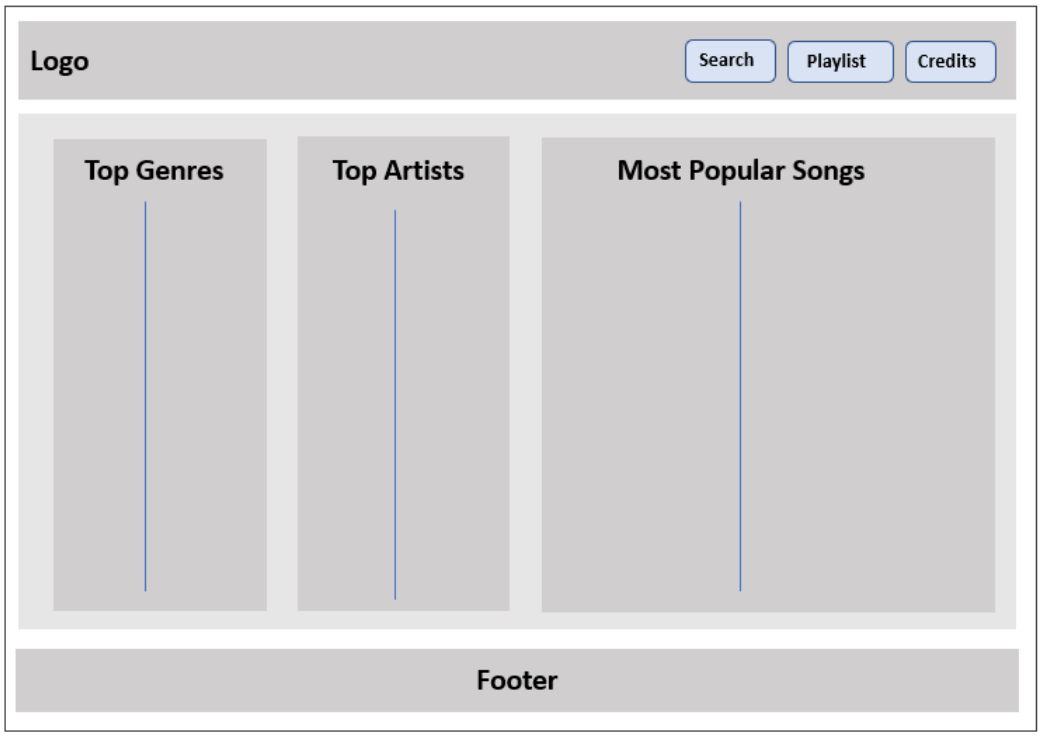
10. **Playlist View**. This view consists of the contents of the playlist. If you want to replicate the sort functionality of your Search/Browse page, go ahead: I don't however require sorting here. The user should be able to remove any song from the playlist. As well, there should be a way to clear the playlist. Also display some summary info about the playlist (i.e., the number of songs in the list and the average popularity).

    For each song in the list, display the song title, artist name, year, genre name, and popularity number. Style the song title so it looks like a hyperlink or style the table row and the cursor so each song in the table is clickable. When the user clicks on a song or title, the view should change to **Single Song View**.



11. **Home View**. This view should be the first to be display to the user. It must contain three blocks:

- Top Genres. Display the top 15 genres based on the number of songs in the data with that genre. Each genre name should be a link that will display the **Search/Browse Songs** view displaying the songs for that genre (as if the user had chosen that genre in the search).
- Top Artists. Display the top 15 artists based on the number of songs in the data by that artist. Each artist name should be a link that will display the **Search/Browse Songs** view displaying the songs for that artist (as if the user had chosen that artist in the search).
- Most Popular Songs. Display the top 15 songs based on their popularity value. Each song name should be a link that will display the **Single Song** view displaying that song.



Testing

Every year students lose many easy marks because they didn't read the requirements carefully enough. When your assignment is getting close to done, I would recommend going through each requirement step and carefully evaluate whether your assignment satisfies each specified requirement.

Submitting and Hosting

Your assignment source code must reside on GitHub and reside on a working public server (in this case GitHub Pages).

GitHub Pages works in conjunction with git and github. You push your html/css/images to github repo; and then push to the host. The instructions for doings so can be found at:

`https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site`

It is up to you whether you want your pages to be public (available to the world) or private (available to only those with access to your github repo).

If you are using a private repo, you must add me as a collaborator!

I would strongly recommend getting your hosting to work a few days before the due date. It's okay if your assignment is still not complete at that point: the idea here is to make sure hosting works ahead of time!

When your hosting is working and the assignment is ready to be marked, then send me an email with the following information:

- The URL of the home page of the site on github pages.
- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.

Hints

1. Begin by consuming the API in the browser. Simply copy and paste the URL into the browser address bar. The JSON can be hard to understand because it doesn't have white space, so use an online JSON formatter (such as https://jsonformatter.curiousconcept.com) via copy and paste to see the JSON structure in an easier to understand format. Alternately, install a json viewer extension in Chrome.
2. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `Energy` and it won't work because the JSON field is actually `energy`.
3. Your `index.html` file will include the markup for all three views. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for the three views.
4. Most of your visual design mark will be determined by how much effort you took to make the views look well designed. Simply throwing up the data with basic formatting will earn you minimal design marks.
5. Most years, students tend to get low marks on the design side of the assignment. I would recommend looking at other sites on the internet and examine how they present data. Notice the use of contrast (weight, color, etc) and spacing.
6. Most of your programming design mark will be based on my assessment of your code using typical code review criteria. For instance, did you modularize your code using functions? Are your functions too long? Is the code documented? Do you have a lot of code duplication (you shouldn't … if you are copy+pasting code, that should tell you that you are doing things wrong from a design standpoint)? Did you make use of object-oriented techniques? Are variables and functions sensibly named? Is your code inefficient (e.g., fetching the same data repeatedly)? Are you using outdated JavaScript techniques (e.g., inline coding, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, etc?