WEB222 Assignment 3

Overview

This assignment is designed to have you practice working with HTML and the DOM in order to create both *static* and *dynamic* web content.

You will prototype a fictional Music App. Your app will focus on a specific music genre and allow users to browse different artists and find specific songs. Because your app's artists and songs will change frequently, we often separate our data from its UI representation. This allows us to quickly make changes and have the app always use the most current music catalogue.

NOTE: in a real music app, our data would be stored in a database. We will simulate working with a database by using JavaScript Objects and Arrays in separate files.

Pick Your Music Genre and Artists, Songs

You need to decide on the following details for your app:

- **Music Genre**: your music app is going to focus on a particular type of music.

- **Name**: what is your app called? Pick something unique and relevant to your chosen music genre.

- **Slogan**: what is your app's slogan (i.e., a short description)? This will help a user to determine if your music app is worth using.

- **Artists**: which music artists/bands does your app include? Based on your chosen music genre, pick **at least 3 artists** that make this type of music. No two students can use the exact same set of artists.

- **Songs**: pick a **minimum of 20 songs** by your chosen artists. Each song will belong to one of the artists.

Modelling your App's Data

Your **artist** and **song** data will go in the files `src/artists.js` and `src/songs.js` respectively. Here is how you should structure these files.

Artists

The `src/artists.js` file will contain an Array of Objects representing an artist. Each artist Object needs three things:

- **artistId**: a unique **String** that identifies this artist. If two artists have the same name, we need a way to tell them apart. In this assignment, we'll use the following format for all artistIds: `/^AID-\d+$/` For example: "AID-5678" or "AID-123451". The start of every artistId will be "AID-" and will then be followed by 1 or more digits.

- **name**: a human-readable **String** meant for display. While the **artistId** is a unique key for the data used by programs, the name is meant to be shown to a user.

- **links**: an **Array** of **Objects** containing one or more links for this artist. An artist might have their own web site, social media, Wikipedia page, etc. Make sure each artist has **at least 2 links**. Each Object in this Array will have the following properties:

    - **url** – a **String** with a URL to the web page (e.g., an Instagram URL)
    - **name** – a **String** with a human-readable name for what the link is (e.g., "Instagram")

Here's an example of what an artist might look like:

```
{
  artistId: "AID-51234",
  name: "Artist Name",
  links: [
    { url: "https://link-to-website.com", name: "Website" },
    { url: "https://instagram.com/name", name: "Instagram" }
  ]
}
```

Songs

Songs are also stored as an Array of Objects, where each song Object needs the following:

- **songId**: a unique **String** that identifies this song. The format is `/^SID-\d+$/` For example: "SID-1234" or "SID-712343". The start of every songId will be "SID-" and will then be followed by 1 or more digits.

- **artistId**: a **String** that indicates which artist this song belongs to (e.g., “AID-51234”), which must match an existing artist.

- **title**: a short **String** that names the song

- **released**: a **Number** with the year (4 digits) that the song was released (e.g., 2020)

- **duration**: a **Number** of seconds (i.e., an integer value) for the song’s length. When we store time values, we often store them in whole numbers (seconds) vs. floats, and convert them to minutes and seconds in the UI for display.

- **mediaUrl**: a **String** with a URL to a site where the user can listen to the song. This might be YouTube or SoundCloud or another site that hosts it.

- **explicitLyrics**: a **Boolean** that indicates whether this song has explicit lyrics. Make sure **at least 2** of your songs have this set to **true**.

Here’s an example of what a song might look like:

```
{
  songId: “SID-13423453”,
  artistId: “AID-51234”,
  title: “Song Title”,
  released: 2020,
  duration: 196,
  mediaUrl: “https://www.youtube.com/watch?v=dQw4w9WgXcQ”,
  isExplicit: true
}
```

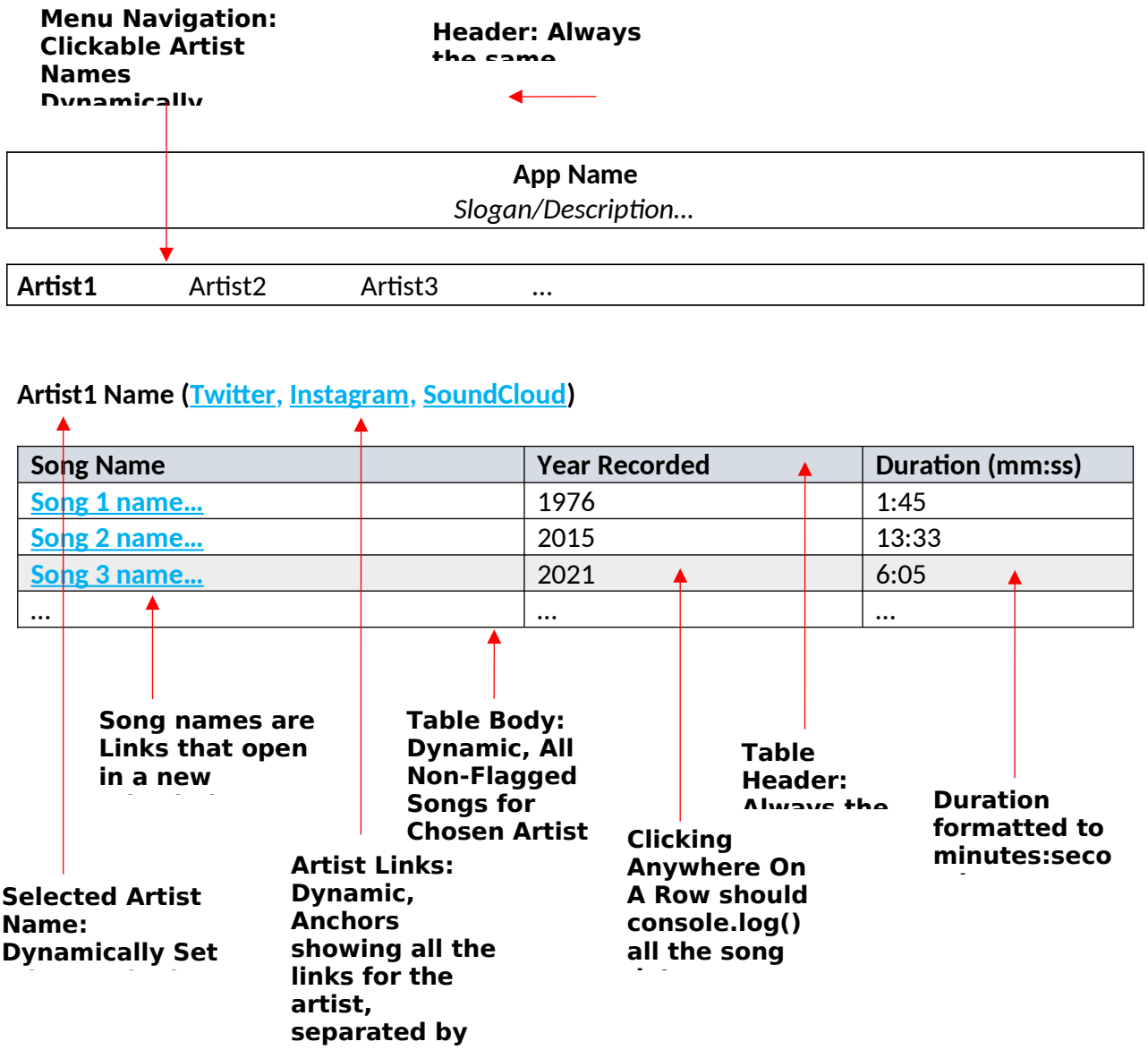
Take some time now to enter all of your app’s data.

App Web Site HTML

Your app’s HTML file is located in `src/index.html`. A brief HTML skeleton has been created, and you are asked to fill in the rest using your information above.

Some of your site will be static (i.e., coded in HTML directly in index.html) and never change. Other parts of the site will be dynamic (i.e., created using DOM API calls at run-time) and will update in response to various events and user actions.

Here is a basic wireframe of what your site needs to include, and which parts are static or dynamic. NOTE: don’t worry too much about how it looks. Focus on the structure and functionality.


Menu Navigation: Clickable Artist Names Dynamically
Header: Always the same
App Name
Slogan/Description...
Artist1 Artist2 Artist3 ...
Artist1 Name (Twitter, Instagram, SoundCloud)
Song Name
Year Recorded
Duration (mm:ss)
Song 1 name...
1976
1:45
Song 2 name...
2015
13:33
Song 3 name...
2021
6:05
...
...
...
Song names are Links that open in a new
Table Body: Dynamic, All Non-Flagged Songs for Chosen Artist
Table Header: Always the
Duration formatted to minutes:seco
Selected Artist Name: Dynamically Set
Artist Links: Dynamic, Anchors showing all the links for the artist, separated by
Clicking Anywhere On A Row should console.log() all the song

Dynamic Content

All of your app's dynamic content will be written in JavaScript in the `src/app.js` file. Here is a list of the tasks you need to complete:

1. Create an event handler to run when the page is loaded. Make sure you don't do anything to the DOM until it's fully loaded. Your function should do the following:
    a. Create all of the buttons for your app's Artists
        i. Loop through all of your Artist objects and create a <button> element for each, adding it to the <nav id="menu">...</nav>
        ii. Use each Artist's name for the button's text
        iii. When the button is clicked, show that Artist's Name, Links, and Songs. See below for more details.
    b. Show a list of Songs in the <tbody>...</tbody> of your Table. By default, you should use your first Artist on load. See below for more details
2. Write a function that will show a list of songs in the <tbody>...</tbody> based on the chosen Artist:
    a. Update the text of the Selected Artist above your table with the Artist's Name and create anchor elements for all of the Artists Links (i.e., you should be able to open these links to see more info about the artist)
    b. Clear the current <tr>...</tr> rows from the <tbody>...</tbody>. HINT: innerHTML = ""
    c. Filter your Songs Array (i.e., use Array.prototype.filter()) to get all Songs for the chosen Artist. HINT: use Array.prototype.includes()
    d. Loop (use Array.prototype.forEach()) over your filtered song list and add them to the table's body using DOM methods (i.e., not innerHTML):
        i. Create a <tr> element
            1. Add a click handler to your <tr> that will console.log() all the song info whenever the user clicks it
        ii. Create <td> elements for the song's name, year, and duration.
        iii. If the song has explicit lyrics, indicate that somehow
        iv. Make the song's title a link to the URL for the song, which opens in a new tab or window
        v. Convert the duration in seconds to a value in mintues:seconds (e.g., 120 seconds would become 2:00)
        vi. Append these <td> elements to the <tr>
        vii. Append this <tr> to the <tbody>

In your solution, you will likely require all of the following:

- console.log() and NOTE that you can log Objects like so: console.log({ object })
- document.querySelector() to find elements in the DOM
- document.createElement() to create new DOM elements
- node.appendChild() to add an element as a child of a DOM node

- element.innerHTML to modify the HTML content of an element. Don't overuse this!

Coding:

Use the website starter project in the assignment ZIP file. Install all dependencies by running the following command in the root of the assignment (e.g., in the same directory as package.json):

npm install

Your code should all be placed in the src/ directory.

Running a Web Server:

You can start a local web server to test your code in a browser by running the following command:

npm start

This will start a server on http://localhost:1234, which you can open in your web browser. If you change the code in `src/` and Save the file(s), the web server will update automatically. This is known as hot-reloading, and is useful for development.

To stop the server, use CTRL + C

Submission:

When you are finished, run the following command to create your submission ZIP file:

npm run prepare-submission

This will generate submission.zip, which you can hand in on Blackboard.