

CS385 Mobile Application Development (Lecture 7)

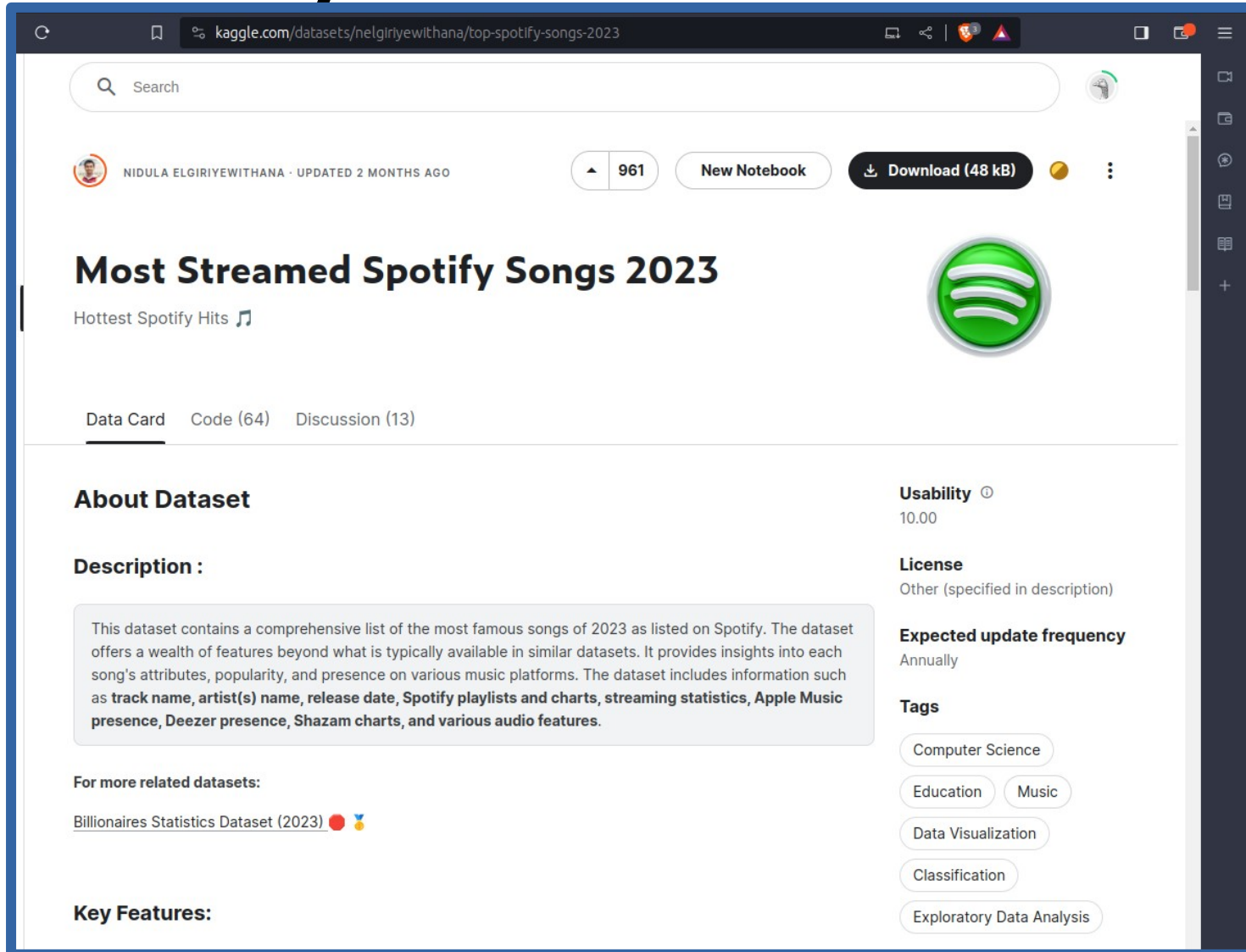


Peter Mooney

In Lecture 7, we want to use a **textbox** to facilitate search

- We want to create a very simple search application. (less than 100 lines of code)
- However, we want to integrate **parent-child communication, event programming** (using the user interface), **useState** hooks, and **props** for data communication between components.
- We'll use an external Javascript file containing a large array of JSON objects.

Spotify search app – from a publicly available dataset



The screenshot shows the Kaggle dataset page for 'Most Streamed Spotify Songs 2023' by Nidula Elgiriye Withana. The page includes a search bar, a user profile, and a 'Download (48 kB)' button. The dataset title is 'Most Streamed Spotify Songs 2023' with the subtitle 'Hottest Spotify Hits'. The page is divided into sections: 'About Dataset', 'Description', 'Usability' (10.00), 'License' (Other), 'Expected update frequency' (Annually), 'Tags' (Computer Science, Education, Music, Data Visualization, Classification, Exploratory Data Analysis), and 'Key Features'. The description states that the dataset contains a comprehensive list of the most famous songs of 2023 as listed on Spotify, offering insights into each song's attributes, popularity, and presence on various music platforms. The dataset includes information such as track name, artist(s) name, release date, Spotify playlists and charts, streaming statistics, Apple Music presence, Deezer presence, Shazam charts, and various audio features. A link to 'Billionaires Statistics Dataset (2023)' is provided for more related datasets.

Search

NIDULA ELGIRIYEWITHANA · UPDATED 2 MONTHS AGO

961 New Notebook Download (48 kB)

Most Streamed Spotify Songs 2023

Hottest Spotify Hits 🎵

Data Card Code (64) Discussion (13)

About Dataset

Description :

This dataset contains a comprehensive list of the most famous songs of 2023 as listed on Spotify. The dataset offers a wealth of features beyond what is typically available in similar datasets. It provides insights into each song's attributes, popularity, and presence on various music platforms. The dataset includes information such as **track name**, **artist(s) name**, **release date**, **Spotify playlists and charts**, **streaming statistics**, **Apple Music presence**, **Deezer presence**, **Shazam charts**, and **various audio features**.

For more related datasets:

[Billionaires Statistics Dataset \(2023\)](#) 🇷🇺 🇺🇦

Key Features:

Usability

10.00

License

Other (specified in description)

Expected update frequency

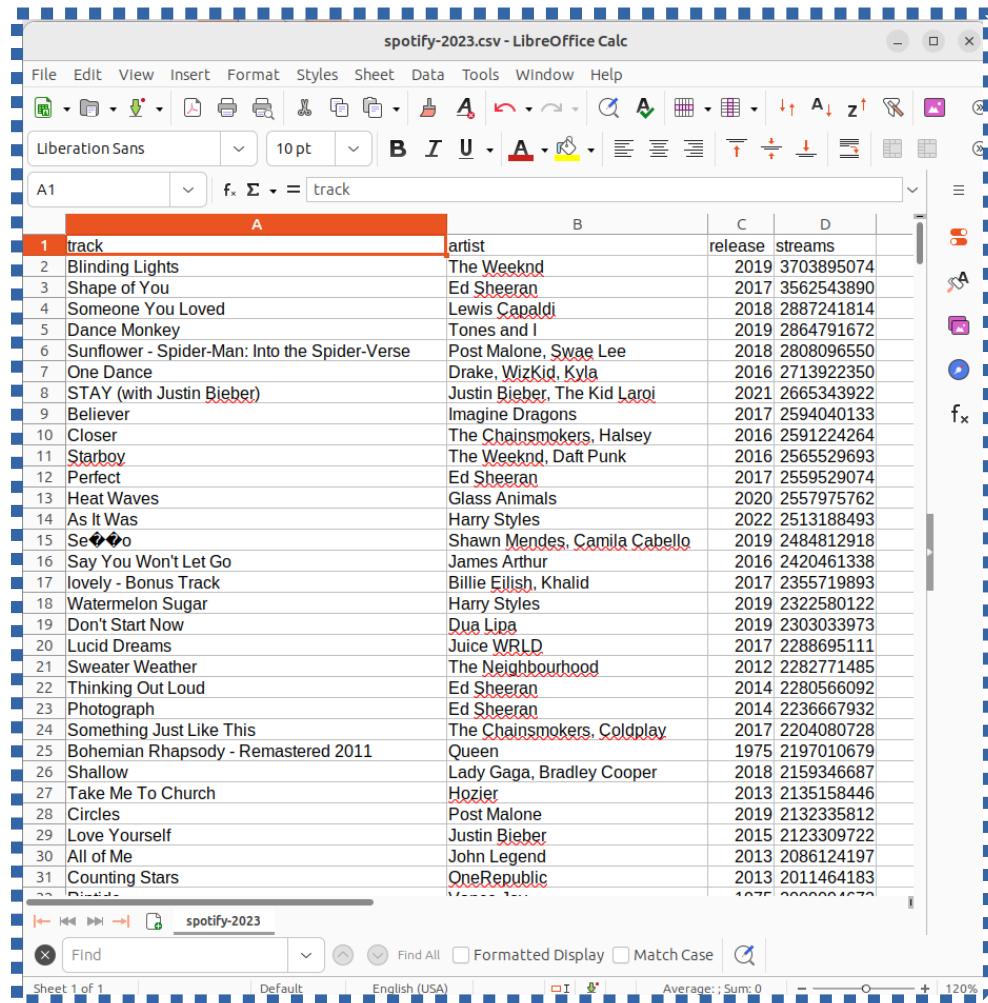
Annually

Tags

Computer Science Education Music Data Visualization Classification Exploratory Data Analysis

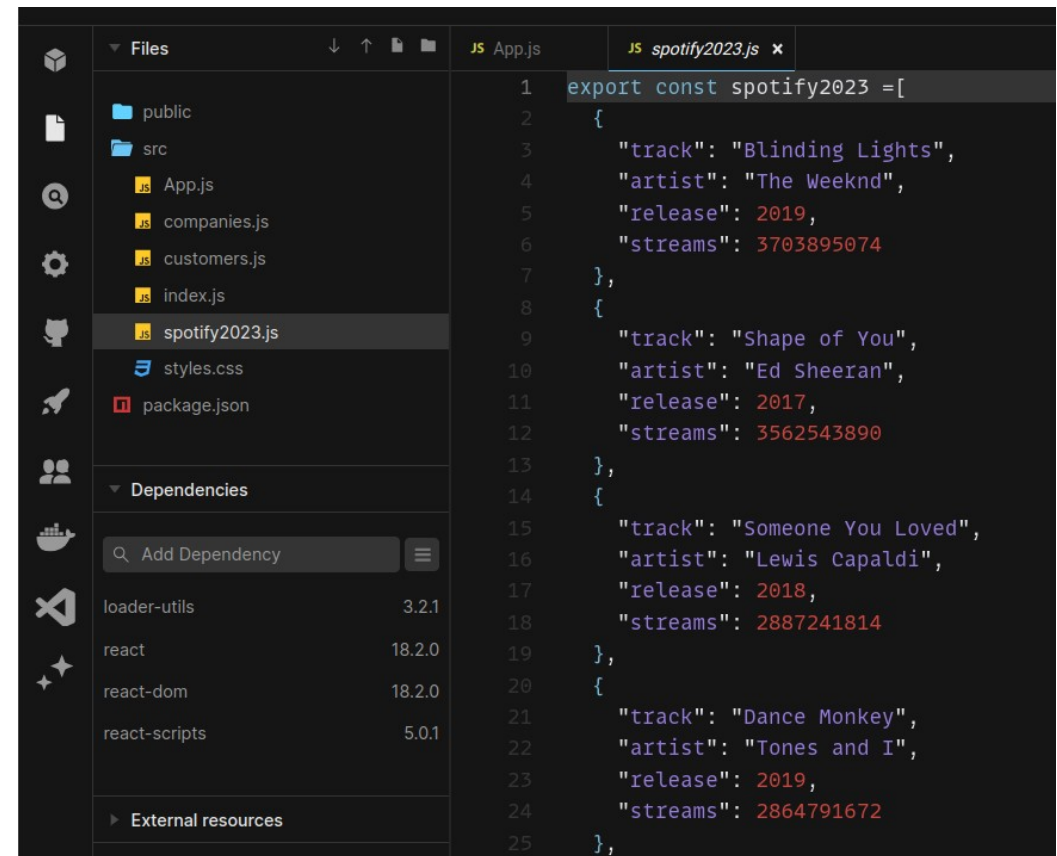
<https://www.kaggle.com/datasets/nelgiriyeewithana/top-spotify-songs-2023>

Pre-development data processing (from CSV to JSON)



spotify-2023.csv - LibreOffice Calc

	A	B	C	D
1	track	artist	release	streams
2	Blinding Lights	The Weeknd	2019	3703895074
3	Shape of You	Ed Sheeran	2017	3562543890
4	Someone You Loved	Lewis Capaldi	2018	2887241814
5	Dance Monkey	Tones and I	2019	2864791672
6	Sunflower - Spider-Man: Into the Spider-Verse	Post Malone, Swae Lee	2018	2808096550
7	One Dance	Drake, WizKid, Kyla	2016	2713922350
8	STAY (with Justin Bieber)	Justin Bieber, The Kid Laroi	2021	2665343922
9	Believer	Imagine Dragons	2017	2594040133
10	Closer	The Chainsmokers, Halsey	2016	2591224264
11	Starboy	The Weeknd, Daft Punk	2016	2565529693
12	Perfect	Ed Sheeran	2017	2559529074
13	Heat Waves	Glass Animals	2020	2557975762
14	As It Was	Harry Styles	2022	2513188493
15	Señorita	Shawn Mendes, Camila Cabello	2019	2484812918
16	Say You Won't Let Go	James Arthur	2016	2420461338
17	lovely - Bonus Track	Billie Eilish, Khalid	2017	2355719893
18	Watermelon Sugar	Harry Styles	2019	2322580122
19	Don't Start Now	Dua Lipa	2019	2303033973
20	Lucid Dreams	Juice WRLD	2017	2288695111
21	Sweater Weather	The Neighbourhood	2012	2282771485
22	Thinking Out Loud	Ed Sheeran	2014	2280566092
23	Photograph	Ed Sheeran	2014	2236667932
24	Something Just Like This	The Chainsmokers, Coldplay	2017	2204080728
25	Bohemian Rhapsody - Remastered 2011	Queen	1975	2197010679
26	Shallow	Lady Gaga, Bradley Cooper	2018	2159346687
27	Take Me To Church	Hozier	2013	2135158446
28	Circles	Post Malone	2019	2132335812
29	Love Yourself	Justin Bieber	2015	2123309722
30	All of Me	John Legend	2013	2086124197
31	Counting Stars	OneRepublic	2013	2011464183
32	Blinding Lights	The Weeknd	2019	3703895074



Files

- public
- src
 - App.js
 - companies.js
 - customers.js
 - index.js
 - spotify2023.js
 - styles.css
 - package.json
- Dependencies
 - loader-utils 3.2.1
 - react 18.2.0
 - react-dom 18.2.0
 - react-scripts 5.0.1
- External resources

```
export const spotify2023 =[
  {
    "track": "Blinding Lights",
    "artist": "The Weeknd",
    "release": 2019,
    "streams": 3703895074
  },
  {
    "track": "Shape of You",
    "artist": "Ed Sheeran",
    "release": 2017,
    "streams": 3562543890
  },
  {
    "track": "Someone You Loved",
    "artist": "Lewis Capaldi",
    "release": 2018,
    "streams": 2887241814
  },
  {
    "track": "Dance Monkey",
    "artist": "Tones and I",
    "release": 2019,
    "streams": 2864791672
  },
  {
    "track": "Sunflower - Spider-Man: Into the Spider-Verse",
    "artist": "Post Malone, Swae Lee",
    "release": 2018,
    "streams": 2808096550
  },
  {
    "track": "One Dance",
    "artist": "Drake, WizKid, Kyla",
    "release": 2016,
    "streams": 2713922350
  },
  {
    "track": "STAY (with Justin Bieber)",
    "artist": "Justin Bieber, The Kid Laroi",
    "release": 2021,
    "streams": 2665343922
  },
  {
    "track": "Believer",
    "artist": "Imagine Dragons",
    "release": 2017,
    "streams": 2594040133
  },
  {
    "track": "Closer",
    "artist": "The Chainsmokers, Halsey",
    "release": 2016,
    "streams": 2591224264
  },
  {
    "track": "Starboy",
    "artist": "The Weeknd, Daft Punk",
    "release": 2016,
    "streams": 2565529693
  },
  {
    "track": "Perfect",
    "artist": "Ed Sheeran",
    "release": 2017,
    "streams": 2559529074
  },
  {
    "track": "Heat Waves",
    "artist": "Glass Animals",
    "release": 2020,
    "streams": 2557975762
  },
  {
    "track": "As It Was",
    "artist": "Harry Styles",
    "release": 2022,
    "streams": 2513188493
  },
  {
    "track": "Señorita",
    "artist": "Shawn Mendes, Camila Cabello",
    "release": 2019,
    "streams": 2484812918
  },
  {
    "track": "Say You Won't Let Go",
    "artist": "James Arthur",
    "release": 2016,
    "streams": 2420461338
  },
  {
    "track": "lovely - Bonus Track",
    "artist": "Billie Eilish, Khalid",
    "release": 2017,
    "streams": 2355719893
  },
  {
    "track": "Watermelon Sugar",
    "artist": "Harry Styles",
    "release": 2019,
    "streams": 2322580122
  },
  {
    "track": "Don't Start Now",
    "artist": "Dua Lipa",
    "release": 2019,
    "streams": 2303033973
  },
  {
    "track": "Lucid Dreams",
    "artist": "Juice WRLD",
    "release": 2017,
    "streams": 2288695111
  },
  {
    "track": "Sweater Weather",
    "artist": "The Neighbourhood",
    "release": 2012,
    "streams": 2282771485
  },
  {
    "track": "Thinking Out Loud",
    "artist": "Ed Sheeran",
    "release": 2014,
    "streams": 2280566092
  },
  {
    "track": "Photograph",
    "artist": "Ed Sheeran",
    "release": 2014,
    "streams": 2236667932
  },
  {
    "track": "Something Just Like This",
    "artist": "The Chainsmokers, Coldplay",
    "release": 2017,
    "streams": 2204080728
  },
  {
    "track": "Bohemian Rhapsody - Remastered 2011",
    "artist": "Queen",
    "release": 1975,
    "streams": 2197010679
  },
  {
    "track": "Shallow",
    "artist": "Lady Gaga, Bradley Cooper",
    "release": 2018,
    "streams": 2159346687
  },
  {
    "track": "Take Me To Church",
    "artist": "Hozier",
    "release": 2013,
    "streams": 2135158446
  },
  {
    "track": "Circles",
    "artist": "Post Malone",
    "release": 2019,
    "streams": 2132335812
  },
  {
    "track": "Love Yourself",
    "artist": "Justin Bieber",
    "release": 2015,
    "streams": 2123309722
  },
  {
    "track": "All of Me",
    "artist": "John Legend",
    "release": 2013,
    "streams": 2086124197
  },
  {
    "track": "Counting Stars",
    "artist": "OneRepublic",
    "release": 2013,
    "streams": 2011464183
  },
  {
    "track": "Blinding Lights",
    "artist": "The Weeknd",
    "release": 2019,
    "streams": 3703895074
  }
]
```

The CS385 Spotify Search app

The screenshot shows a CodeSandbox environment with the following components:

- File Explorer:** Lists files including `public`, `src`, `App.js`, `companies.js`, `customers.js`, `index.js`, `spotify2023.js` (selected), `styles.css`, and `package.json`.
- Dependencies:** Lists installed packages: `loader-utils` (3.2.1), `react` (18.2.0), `react-dom` (18.2.0), and `react-scripts` (5.0.1).
- Code Editor:** Displays the `spotify2023.js` file with the following JavaScript code:

```
213 track: "good 4 u",
214 "artist": "Olivia Rodrigo",
215 "release": 2021,
216 "streams": 1887039593
217 },
218 {
219   "track": "drivers license",
220   "artist": "Olivia Rodrigo",
221   "release": 2021,
222   "streams": 1858144199
223 },
224 {
225   "track": "Demons",
226   "artist": "Imagine Dragons",
227   "release": 2012,
228   "streams": 1840364617
229 },
230 {
231   "track": "Lose Yourself - Soundtrack Version",
232   "artist": "Eminem",
233   "release": 2002,
234   "streams": 1829992958
235 },
236 {
237   "track": "INDUSTRY BABY (feat. Jack Harlow)",
238   "artist": "Jack Harlow, Lil Nas X",
239   "release": 2021,
240   "streams": 1814349763
241 },
242 {
243   "track": "Another Love",
244   "artist": "Tom Odell",
245   "release": 2012,
```
- Browser Window:** Displays the running application at `https://7gr67n.csb.app/`. The UI includes:
 - Header:** "Parent component CS385 Spotify Search"
 - Text:** "Your current search term is [IMAGINE]"
 - Form:** "Type your search here:" with an input field containing "IMAGINE".
 - Section:** "Child Component: Search Results"
 - Text:** "There are 5 search results"
 - Results:**
 - Imagine Dragons, *Believer* Streams: 2594040133 (At least 2 billion!)
 - Imagine Dragons, *Demons* Streams: 1840364617 (At least 1 billion!)
 - Imagine Dragons, *League of Legends, JID, Arcane, Enemy (with JID)* - from the series *Arcane League of Legends* Streams: 1223481149 (At least 1 billion!)
 - Imagine Dragons, *League of Legends, Arcane, Enemy* - from the series *Arcane League of Legends* Streams: 1223481149 (At least 1 billion!)
 - Imagine Dragons, *Bones* Streams: 838079900 (At least 8 hundred million!)
- Console:** Shows "Console was cleared".

Lecture 7 will consider this applicatoin development in 5 parts

- **Part 1** – implementing a Search Box and storing the typed data into a state variable
- **Part 2** – Using a child component SPECIFICALLY to process and display the results of the user search
- **Part 3**– Using our own filter function within a child component for searching
- **Part 4** – conditional rendering of output on the user interface (new for Lecture 7)
- **Part 5** – writing our own helper functions (basic example of writing a function for specific task)

- **Part 1** – implementing a Search Box and storing the typed data into a state variable
- **Part 2** – Using a child component SPECIFICALLY to process and display the results of the user search
- **Part 3**– Using our own filter function within a child component for searching
- **Part 4** – conditional rendering of output on the user interface (new for Lecture 7)
- **Part 5** – writing our own helper functions (basic example of writing a function for specific task)

```

31 // Here is our textbox handler function.
32 // This handles the event that is fired when
33 // the search form (text box) changes
34 function onSearchFormChange(event) {
35     // An event is generated by Javascript.
36     // We use the hook setSearchTerm to safely assign
37     // the current value in the textbox to searchTerm
38     setSearchTerm(event.target.value);
39 }
40
41 return (
42     <>
43     <h1>Parent component CS385 Spotify Search</h1>
44     <p>Your current search term is [{searchTerm}]</p>
45     <form>
46         <h3>Type your search here: </h3>
47         <input onChange={onSearchFormChange} type="text" />
48     </form>
49     <hr />
50     <ResultsComponent
51         searchTermFromParent={searchTerm}
52         spotifyArrayFromParent={spotify2023}
53     />
54 </>
55 );
56 }
57

```

Browser Tests
https://7gr67n.csb.app/

Parent component CS385 Spotify Search

Your current search term is [The]

Type your search here: **1**

Child Component: Search Results **3**

There are 39 search results **2**

LOTS OF RESULTS **4**

The Weeknd, *Blinding Lights* Streams: 3703895074 (At least 3 billion!)

Post Malone, Swae Lee, *Sunflower - Spider-Man: Into the Spider-Verse* Streams: 2808096550 (At least 2 billion!)

Justin Bieber, The Kid Laroi, *STAY (with Justin Bieber)* Streams: 2665343922 (At least 2 billion!)

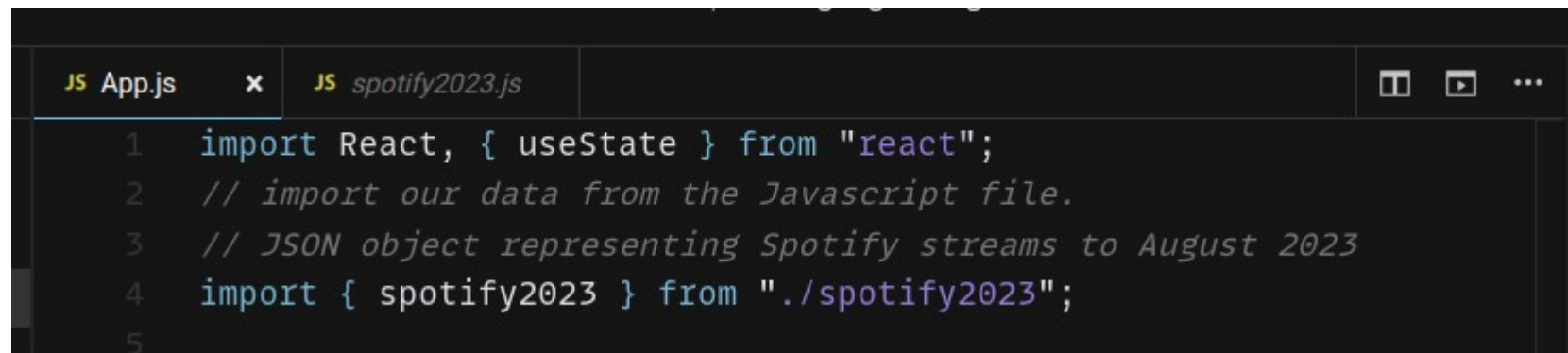
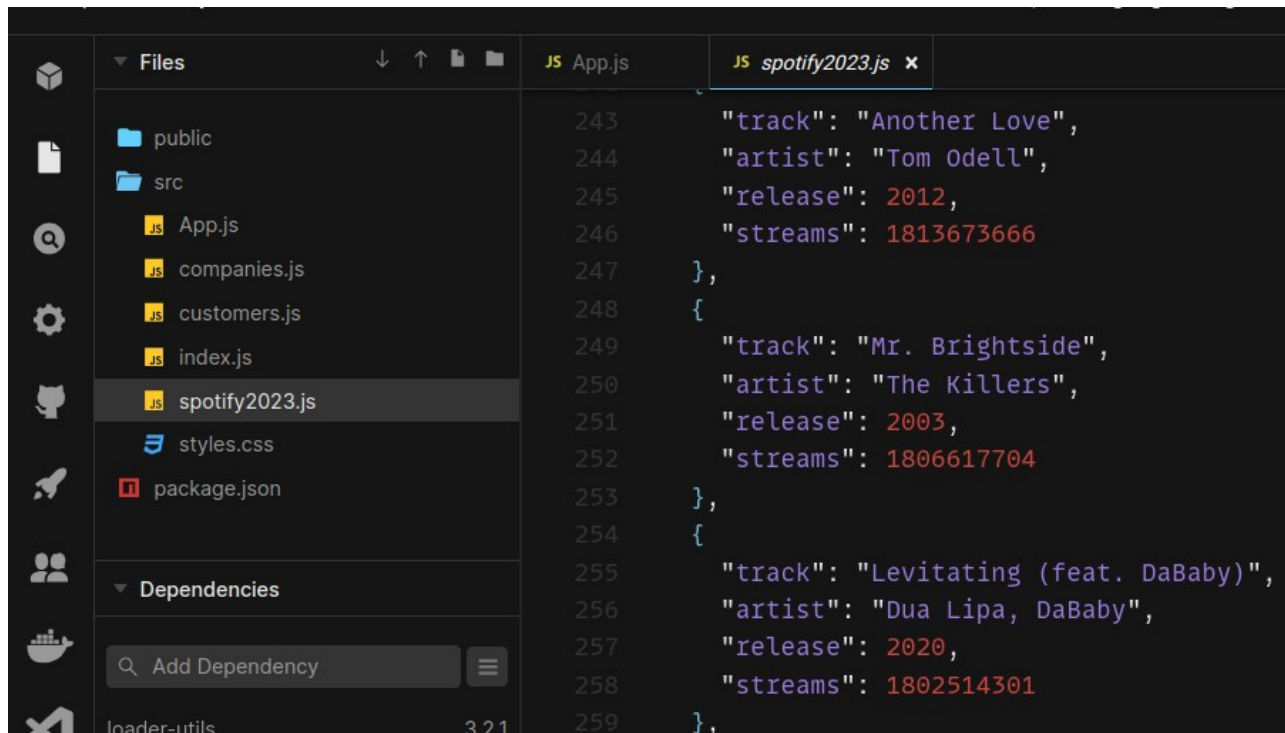
The Chainsmokers, Halsey, *Closer* Streams: 2591224264 (At least 2 billion!)

The Weeknd, Daft Punk, *Starboy* Streams: 2565529693 (At least 2 billion!)

The Neighbourhood, *Sweater Weather* Streams: 2282771485 (At least 2 billion!) **5**

Console Problems React DevTools Filter All
Console was cleared

Part 0: Let's start by importing our array of Spotify stream objects



Part 1: - use a textbox in the App component

- Line 47 declares our text or input box.
- Line 29 declares our **searchTerm** and the set method for this **state variable**.
- Crucially, line 34 declares a function called **onSearchFormChange** which is **TRIGGERED every time there is a CHANGE within the text box** (that is - everytime someone types something, even one character)

```
25 // Parent component - App
26 function App() {
27   // lets keep the searchTerm as a state variable
28   // searchTerm is given an initial value of empty string
29   const [searchTerm, setSearchTerm] = useState("");
30
31   // Here is our textbox handler function.
32   // This handles the event that is fired when
33   // the search form (text box) changes
34   function onSearchFormChange(event) {
35     // An event is generated by Javascript.
36     // We use the hook setSearchTerm to safely assign
37     // the current value in the textbox to searchTerm
38     setSearchTerm(event.target.value);
39   }
40
41   return (
42     <>
43     <h1>Parent component CS385 Spotify Search</h1>
44     <p>Your current search term is [{searchTerm}]</p>
45     <form>
46       <h3>Type your search here: </h3>
47       <input onChange={onSearchFormChange} type="text" />
48     </form>
```

Part 1 – handling events in Javascript.

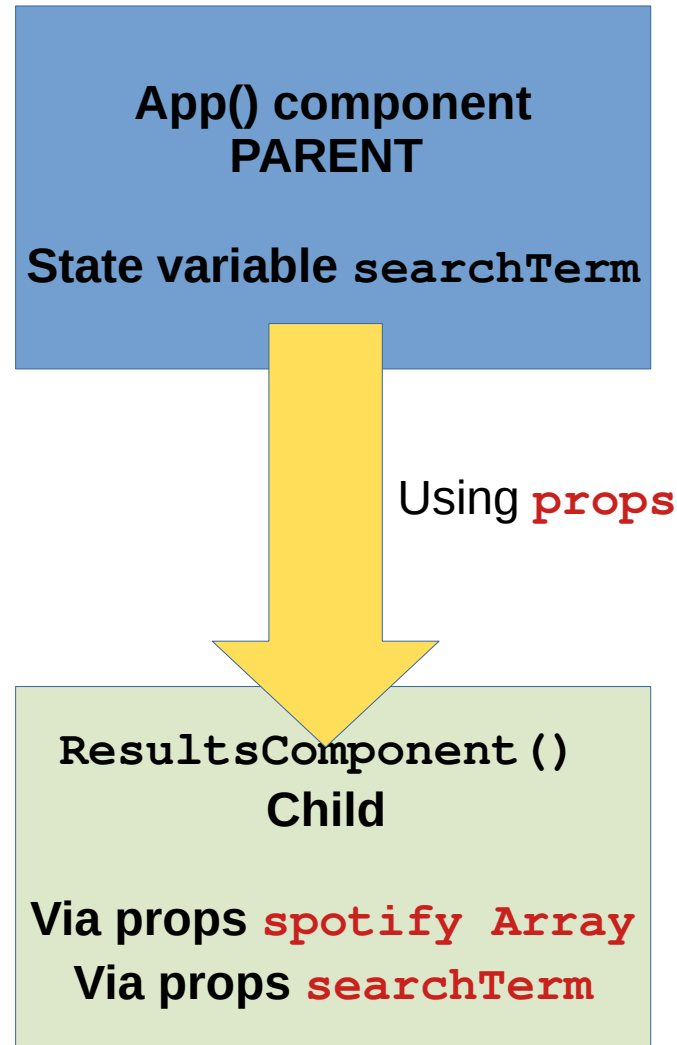
As the event happens, the value is stored in **searchTerm**

- Every time the user makes a change in the text box (types something) an EVENT is triggered.
- Javascript then has access to that EVENT Object.
- By looking at the properties of the event object (line 38) – **target.value** – we can obtain what the user has typed in!
- So we are constantly updated on what the user has typed in!

```
25 // Parent component - App
26 function App() {
27   // lets keep the searchTerm as a state variable
28   // searchTerm is given an initial value of empty string
29   const [searchTerm, setSearchTerm] = useState("");
30
31   // Here is our textbox handler function.
32   // This handles the event that is fired when
33   // the search form (text box) changes
34   function onSearchFormChange(event) {
35     // An event is generated by Javascript.
36     // We use the hook setSearchTerm to safely assign
37     // the current value in the textbox to searchTerm
38     setSearchTerm(event.target.value);
39   }
40
41   return (
42     <>
43     <h1>Parent component CS385 Spotify Search</h1>
44     <p>Your current search term is [{searchTerm}]</p>
45     <form>
46       <h3>Type your search here: </h3>
47       <input onChange={onSearchFormChange} type="text" />
48     </form>
```

Part 2 – using the child component to render or display results

- The child component called **ResultsComponent** will need to (a) **filter the array** using the **search term** and then (b) **display or render the filtered objects** using a map function.
- This **COMPONENT-based** approach places **ALL OF THE RESPONSIBILITY** for rendering results into the **CHILD COMPONENT**. The parent component is just responsible for capturing the **searchTerm**.



Part 2 – Parent-child communication using **props**

```
41  return (  
42    <>  
43      <h1>Parent component CS385 Spotify Search</h1>  
44      <p>Your current search term is [{searchTerm}]</p>  
45      <form>  
46        <h3>Type your search here: </h3>  
47        <input onChange={onSearchFormChange} type="text" />  
48      </form>  
49      <hr />  
50      <ResultsComponent  
51        searchTermFromParent={searchTerm}  
52        spotifyArrayFromParent={spotify2023}  
53      />  
54    </>  
55  );  
56 }
```



Using **props**



- On Line 50 – the parent uses props to pass the **searchTerm** and the **spotify2023** array to the child component called **ResultsComponent**

Part 3 – using a filter function WITHIN the child component

- With the **ResultsComponent** – we will need to filter the spotify array based on the **searchTerm** value.
- Recall our previous filter functions. This time we want to filter based on the **artist** and the **track** properties of the objects in the **spotify2023** array

```
58 // This is the child component. It is used to display the results
59 // of the searches conducted. The parent must provide the searchTerm
60 // and access to the array of JSON objects.
61
62 function ResultsComponent(props) {
63   // Within this component or function we create our
64   // filter function. It will be needed to conduct
65   // the search of the array of JSON objects.
66
67   function spotifyFilterFunction(searchTerm) {
68     return function (spotifyObject) {
69       // convert everything to lower case for string matching
70       let artist = spotifyObject.artist.toLowerCase();
71       let track = spotifyObject.track.toLowerCase();
72       return (
73         searchTerm !== "" &&
74         (track.includes(searchTerm.toLowerCase()) ||
75          artist.includes(searchTerm.toLowerCase()))
76       );
77     };
78   }
```

```
321   "track": "Smells Like Teen Spirit - Remastered 2021",
322   "artist": "Nirvana",
323   "release": 1991,
324   "streams": 1690192927
325 },
326 {
327   "track": "Without Me",
328   "artist": "Eminem",
329   "release": 2002,
330   "streams": 1687664027
331 },
332 {
333   "track": "When I Was Your Man",
334   "artist": "Bruno Mars",
335   "release": 2012,
336   "streams": 1661187319
337 },
```

Part 3- we now use the `filter()` function to search the array

- We shall also use the `filter()` function to give us the number of SEARCH RESULTS (objects that passed through the filter)
- We'll use the `filter` function with the map function to display or render the results of the current search (based on the value of the `searchTerm`)
- This is a very standardised approach to searching and results display in applications.

Part 3- we now use the `filter()` function to search the array

```
79 // We can use the filter function to tell us how many search results
80 // we have. We find the length of the filtered array
81
82 let numberResults = props.spotifyArrayFromParent.filter(
83   spotifyFilterFunction(props.searchTermFromParent)
84 ).length;
```

- We can see TWO uses of the `filter` function. On line 82 – the filter function ALWAYS results an array of filtered results. The length of this array is the number of search results.
- On line 93 – 99 we have our standard `map` function `filter` function combination for rendering our results.

```
93 {props.spotifyArrayFromParent
94   .filter(spotifyFilterFunction(props.searchTermFromParent))
95   .map((a, index) => (
96     <p key={index}>
97       <b>{a.artist}</b>, <i>{a.track}</i> Streams: {a.streams}{" "}
98     </p>
99   )
```

Part 4 – conditional rendering

- All rendering we have seen so far involved no decision making.
- **Conditional rendering** allows us to render JSX (Javascript and HTML) based on some condition(s) being **true** or **false**
- It is **VERY** useful for many situations

```
{ condition && <div>Write something</div> }  
  
{ !condition ? <div>Error do it again</div> :  
  <div>Congratulations</div> }
```

Part 4 – conditional rendering

- **Lines 90 – 92 contain conditional rendering** based on the value of the **numberResults** variable. We ONLY render if the condition is true (hence the use of **&&** in the statements)

```
81
82 let numberResults = props.spotifyArrayFromParent.filter(
83   spotifyFilterFunction(props.searchTermFromParent)
84 ).length;
85
86 return (
87   <>
88     <h1>Child Component: Search Results</h1>
89     <h2>There are {numberResults} search results </h2>
90     {numberResults === 0 && <p>No results</p>}
91     {numberResults > 0 && numberResults < 10 && <p>Some results, not many</p>}
92     {numberResults > 10 && <p>Lots of results</p>}
93     {props.spotifyArrayFromParent
94       .filter(spotifyFilterFunction(props.searchTermFromParent))
95       .map((a, index) => (
96         <p key={index}>
97           <b>{a.artist}</b>, <i>{a.track}</i> Streams: {a.streams}{" "}
98           {writeNumberAsWords(a.streams)}
99         </p>
```

Your current search term is [Monkeys]

Type your search here:

Child Component:

There are 3 search results

Some results, not many

Arctic Monkeys, Do I Wanna Know? Streams: 1267333350 (At least 1 billion!)

Arctic Monkeys, I Wanna Be Yours Streams: 1267333350 (At least 1 billion!)

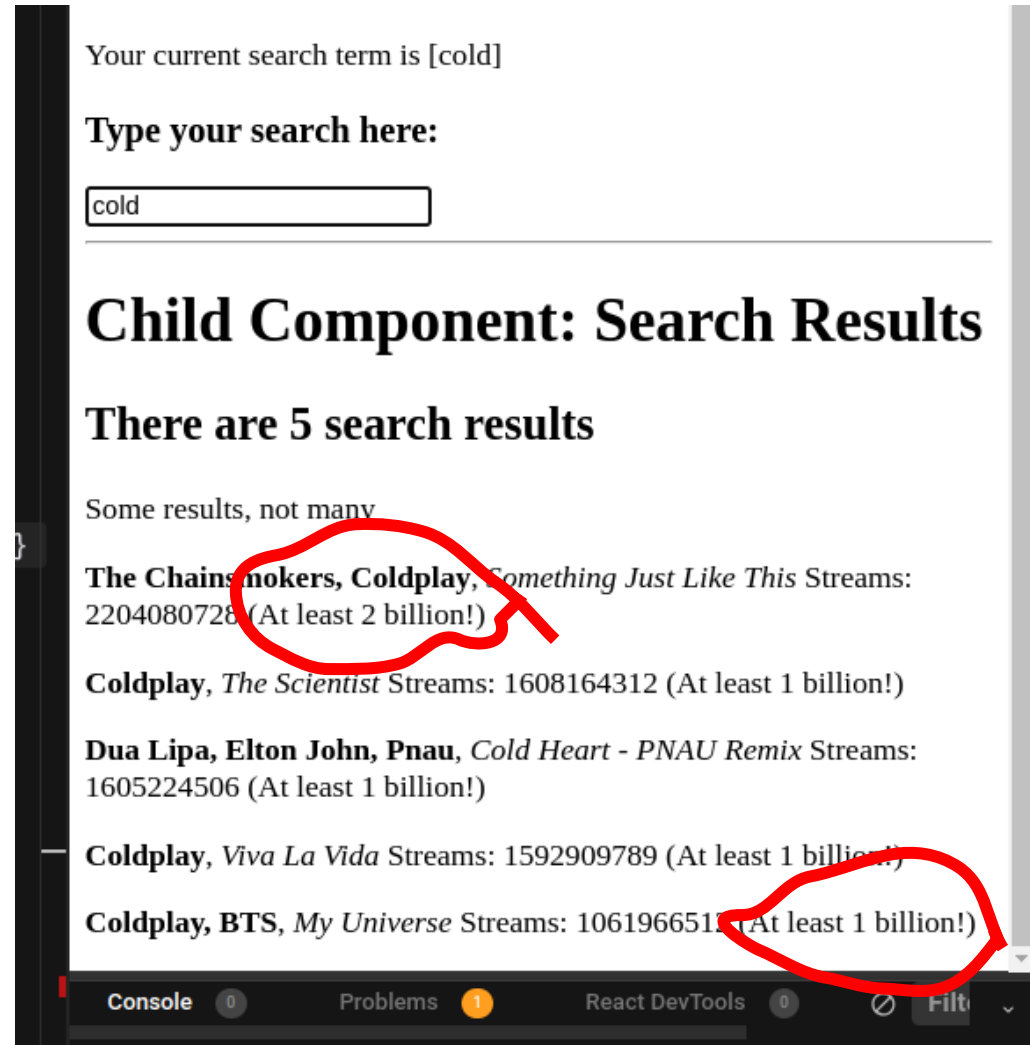
Arctic Monkeys, Why'd You Only Call Me When You're High? Streams: 1267333350 (At least 1 billion!)

Part 5 – writing our own helper functions for specific tasks

- We have used many pre-written Javascript functions up to this point (**map**, **filter**, **includes**, **toUpperCase**, and so on).
- But what happens if we want a specific piece of functionality (and we want to use it many times)?
- We can always take the option of writing our own functions. And, we can use them over and over again. And, we can change and update them easily

Part 5 – our own helper function

- You can see the numerical value for number of spotify streams shown below.
- It would be nice to provide our users with a friendly (non complicated) sentence about how many streams that number represents.
- No Javascript function exists (that I know of) that will provide this output when given a numerical value as input.
- Let's write our own function



Part 5 – our own

writeNumberAsWords function

- (I know!) there are many many ways to write this function.
- The key concept here is to (a) know that we can write our own function and then (b) how to apply it.
- Note – this wouldn't work for all numerical inputs (that's fine for now)

```
6 // Writing our own functions for use in our app
7 // Writing a comment about a specific number.
8 function writeNumberAsWords(n) {
9   let nAsStr = n.toString(10); // our number is in base 10
10  let digits = nAsStr.length; // number of digits
11  let firstDigit = nAsStr.charAt(0);
12
13  let message = "nothing";
14  if (digits === 9) {
15    message = "(At least " + firstDigit + " hundred million!)";
16  } else if (digits === 10) {
17    message = "(At least " + firstDigit + " billion!)";
18  } else {
19    message = "(Lots of streams)";
20  }
21
22  return message;
23 }
```

IMPORTANT – notice that we write this function OUTSIDE of the **parent** and **child** component meaning that, in practice, ANY component could REUSE our function.

Part 5 – using our own **writeNumberAsWords** function

```
5 // Writing our own functions for use in our app
6 // Writing a comment about a specific number.
7
8 function writeNumberAsWords(n) {
9   let nAsStr = n.toString(10); // our number is in base 10
10  let digits = nAsStr.length; // number of digits
11  let firstDigit = nAsStr.charAt(0);
12
13  let message = "nothing";
14  if (digits === 9) {
15    message = "(At least " + firstDigit + " hundred million!)";
16  } else if (digits === 10) {
17    message = "(At least " + firstDigit + " billion!)";
18  } else {
19    message = "(Lots of streams)";
20  }
21
22  return message;
23 }
```

Masked Wolf, Astronaut In The Ocean Streams: 1150474110 (At least 1 billion!)

Bruno Mars, Anderson .Paak, Silk Sonic, Leave The Door Open Streams: 1115880852 (At least 1 billion!)

Keane, Somewhere Only We Know Streams: 1089402494 (At least 1 billion!)

Halsey, BTS, Boy With Luv (feat. Halsey) Streams: 1065580332 (At least 1 billion!)

Tyler, The Creator, Kali Uchis, See You Again Streams: 1047101291 (At least 1 billion!)

Kate Bush, Running Up That Hill (A Deal With God) Streams: 1024858327 (At least 1 billion!)

Sean Paul, Dua Lipa, No Lie Streams: 956865266 (At least 9 hundred million!)

Giveon, HEARTBREAK ANNIVERSARY Streams: 951637566 (At least 9 hundred million!)

Kanye West, Heartless Streams: 887906111 (At least 8 hundred million!)

Riton, Nightcrawlers, Mufasa & Hypeman, Dopamine, Friday (feat. Mufasa & Hypeman) - Dopamine Re-Edit Streams: 863756573 (At least 8 hundred million!)

Aerosmith, Dream On Streams: 838586769 (At least 8 hundred million!)

Surf Curse, Freaks Streams: 824420218 (At least 8 hundred million!)

Frank Ocean, Lost Streams: 822239726 (At least 8 hundred million!)

- We actually use our new function within the **child component**. The child is responsible for displaying our results. Line 98

```
93 {props.spotifyArrayFromParent
94   .filter(spotifyFilterFunction(props.searchTermFromParent))
95   .map((a, index) => (
96     <p key={index}>
97       <b>{a.artist}</b>, <i>{a.track}</i> Streams: {a.streams}{ " "
98       {writeNumberAsWords(a.streams)}
99     </p>
```

Spotify Search example – lessons learned!

- With the omission of a nice UI – the functionality for a searching application takes less than 100 lines of code.
- **We simplified our code by ENCAPSULATING the logic for display/rendering of results into the child componet.**
- **Parent-child communication allowed us to write our code in a more elegant way** and avoid using using one large App0 component



CS385 Mobile Application Development (Lecture 7)



Peter Mooney