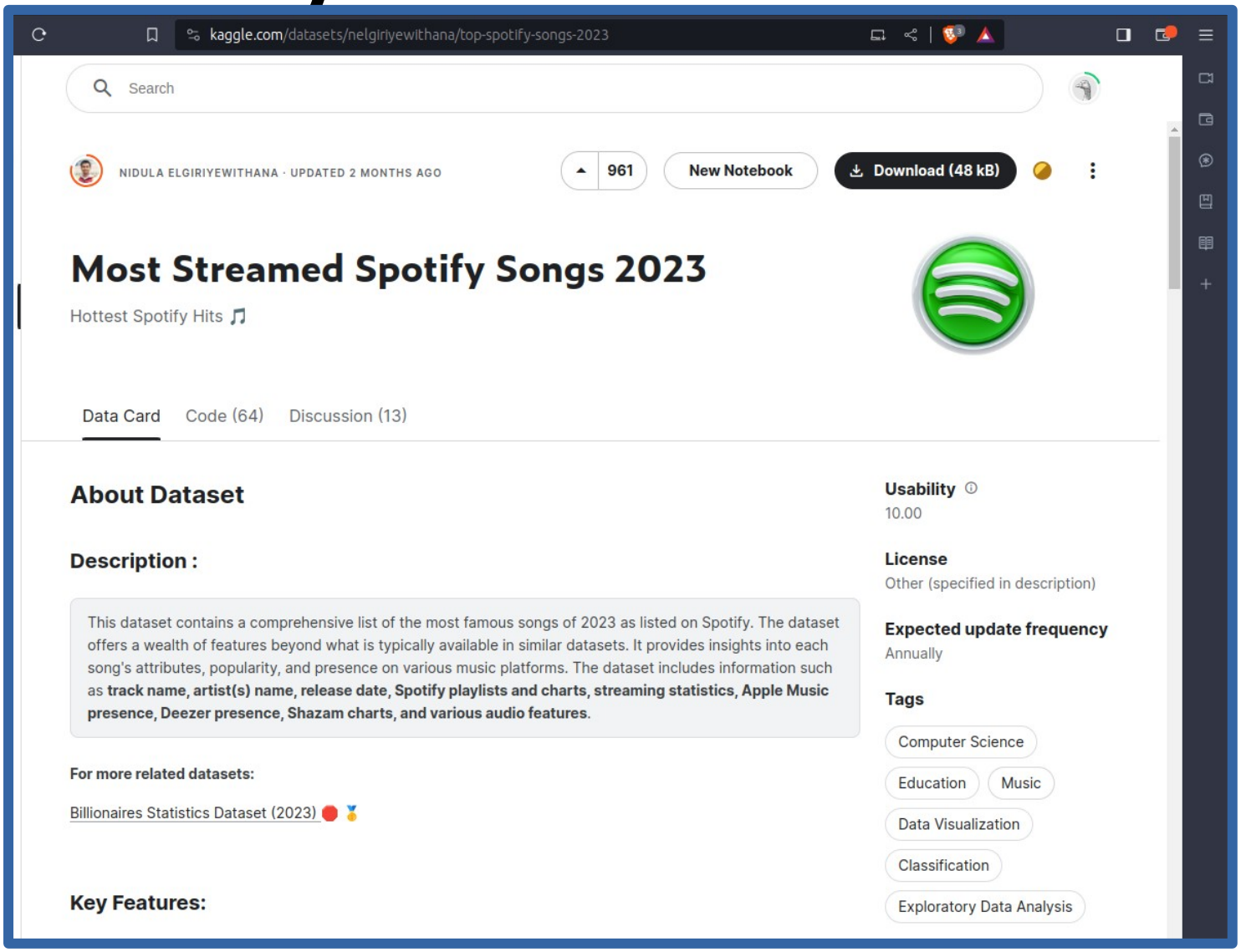
CS385 Mobile Application Development (Lecture 7)

Peter Mooney

In Lecture 7, we want to use a textbox to faciliate search

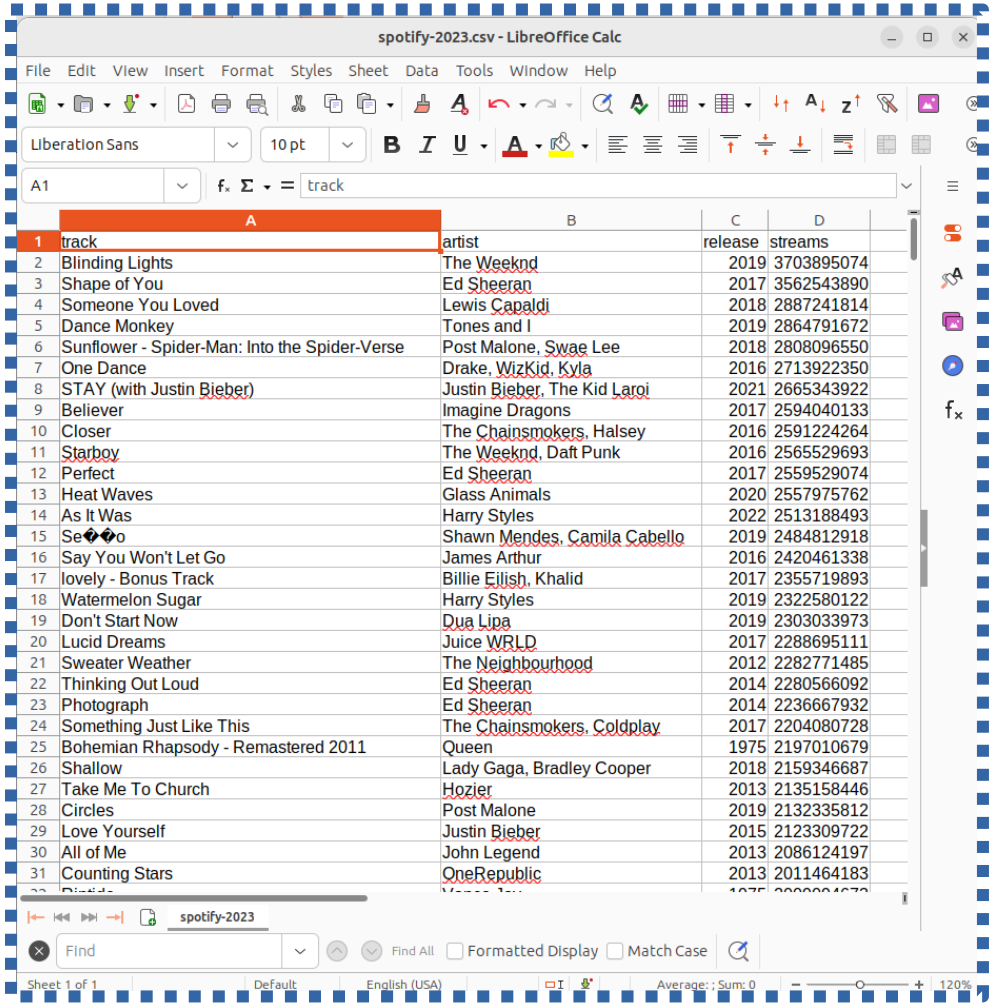
- We want to create a very simple search application. (less than 100 lines of code)
- However, we want to integrate **parent-child communication**, **event programming** (using the user interface), **useState** hooks, and **props** for data communication between components.
- We'll use **an external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

Spotify search app – from a publicly available dataset

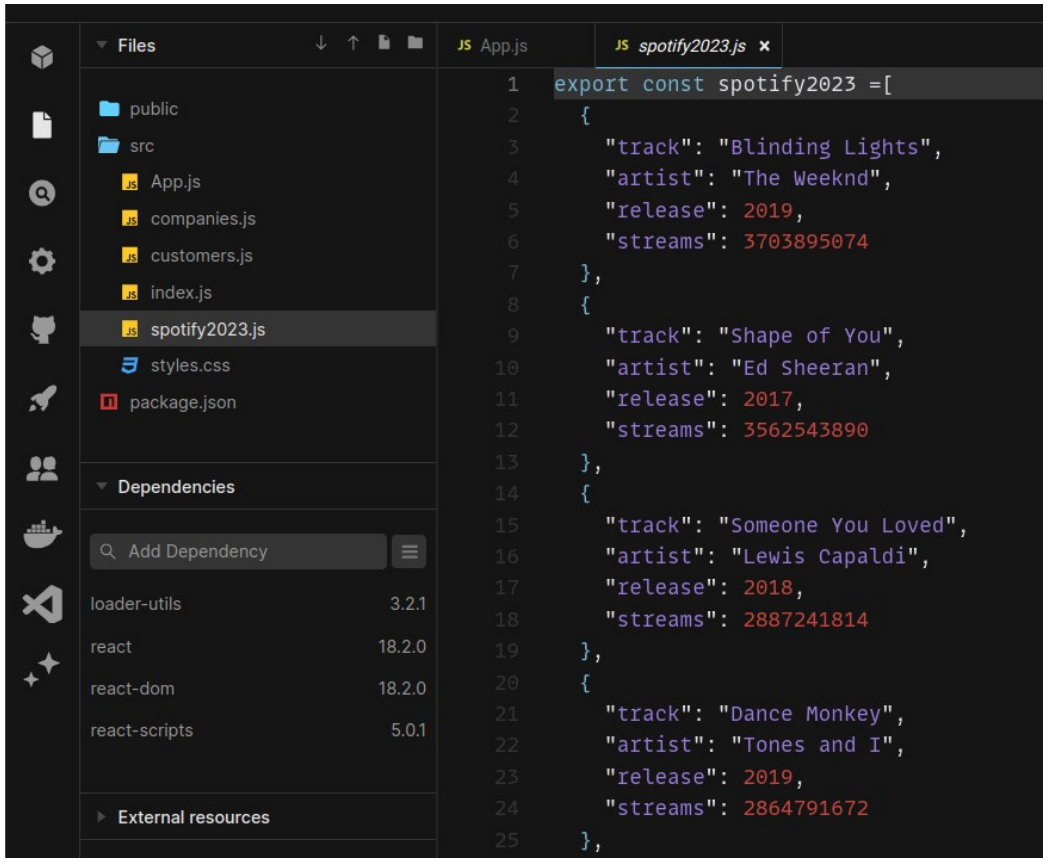


https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023

Pre-development data processing (from CSV to JSON)



	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	Se►►o	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



The CS385 Spotify Search app

codesandbox.io/s/nostalgic-grass-7gr67n?file=/src/spotify2023.js

petermooney / Drafts / nostalgic-grass-7gr67n

Share | Fork | Crea

Files

- public
- src
  - App.js
  - companies.js
  - customers.js
  - index.js
  - spotify2023.js
  - styles.css
- package.json

Dependencies

Add Dependency

loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

External resources

App.js | spotify2023.js

```
    "artist": "Olivia Rodrigo",
    "release": 2021,
    "streams": 1887039593
  },
  {
    "track": "drivers license",
    "artist": "Olivia Rodrigo",
    "release": 2021,
    "streams": 1858144199
  },
  {
    "track": "Demons",
    "artist": "Imagine Dragons",
    "release": 2012,
    "streams": 1840364617
  },
  {
    "track": "Lose Yourself - Soundtrack Version",
    "artist": "Eminem",
    "release": 2002,
    "streams": 1829992958
  },
  {
    "track": "INDUSTRY BABY (feat. Jack Harlow)",
    "artist": "Jack Harlow, Lil Nas X",
    "release": 2021,
    "streams": 1814349763
  },
  {
    "track": "Another Love",
    "artist": "Tom Odell",
    "release": 2012,
```

Browser | Tests

https://7gr67n.csb.app/

Parent component CS385 Spotify Search

Your current search term is [IMAGINE]

Type your search here:

IMAGINE

Child Component: Search Results

There are 5 search 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 Leag*
Streams: 1223481149 (At least 1 billion!)

Imagine Dragons, League of Legends, Arcane, *Enemy - from the series Arcane League of Legends* St
1223481149 (At least 1 billion!)

Imagine Dragons, *Bones* Streams: 838079900 (At least 8 hundred million!)

Console 0 | Problems 1 | React DevTools 0 | Filter

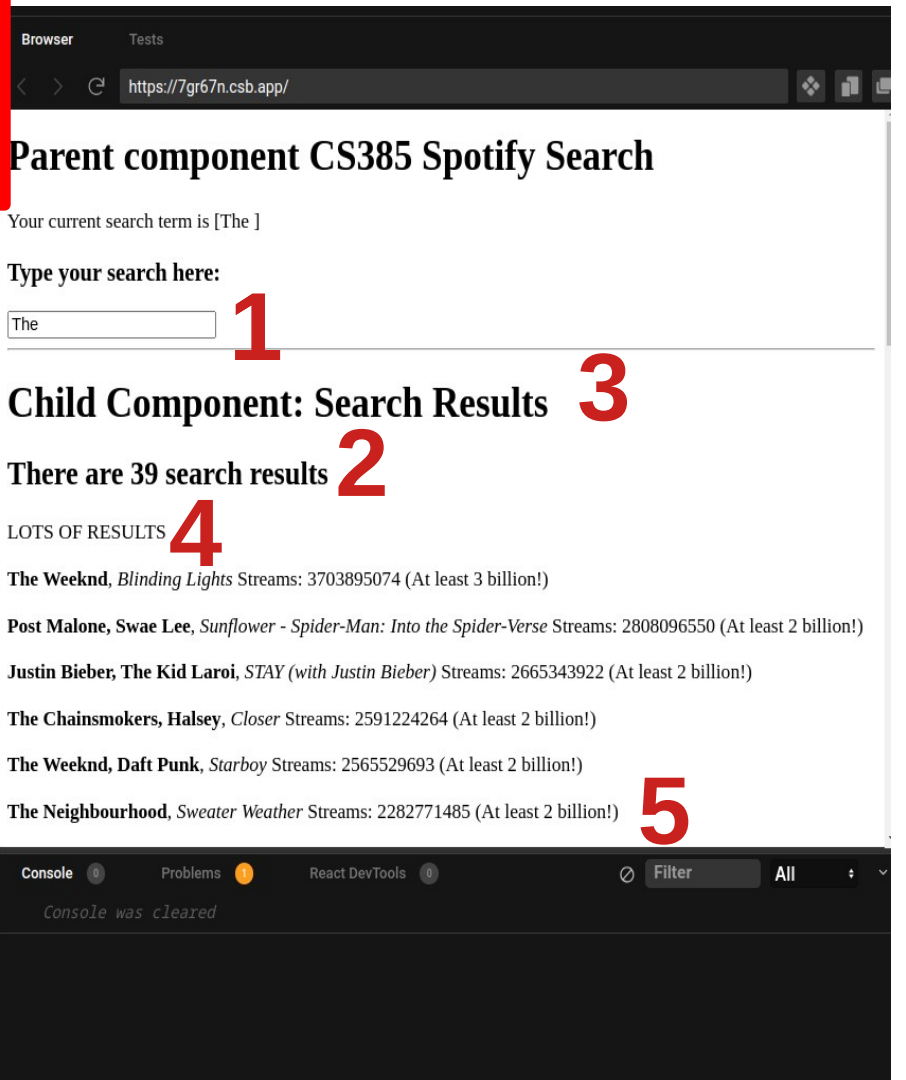
Console was cleared

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="tex
48        </form>
49        <hr />
50        <ResultsComponent
51          searchTermFromParent={searchTerm}
52          spotifyArrayFromParent={spotify2023}
53        />
54      </>
55    );
56  }
57
```



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

Files

- public
- src
  - App.js
  - companies.js
  - customers.js
  - index.js
  - spotify2023.js
  - styles.css
- package.json

Dependencies

Add Dependency

loader-utils 3.2.1

App.js | spotify2023.js

```
243      "track": "Another Love",
244      "artist": "Tom Odell",
245      "release": 2012,
246      "streams": 1813673666
247    },
248    {
249      "track": "Mr. Brightside",
250      "artist": "The Killers",
251      "release": 2003,
252      "streams": 1806617704
253    },
254    {
255      "track": "Levitating (feat. DaBaby)",
256      "artist": "Dua Lipa, DaBaby",
257      "release": 2020,
258      "streams": 1802514301
259    },
```

App.js | spotify2023.js

```
1  import React, { useState } from "react";
2  // import our data from the Javascript file.
3  // JSON object representing Spotify streams to August 2023
4  import { spotify2023 } from "./spotify2023";
5
```

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 varaiable
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>
```

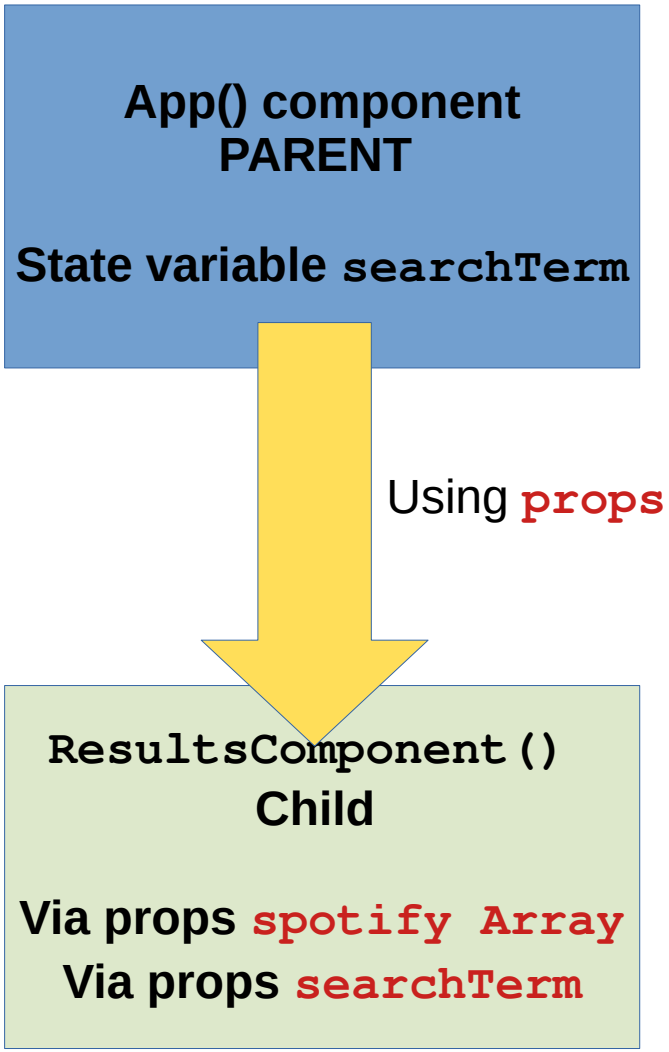
Part 1 – handling events in Javascript. As the event happens, the value is stored in `searchTerm`

- **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 varaiable
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>
```

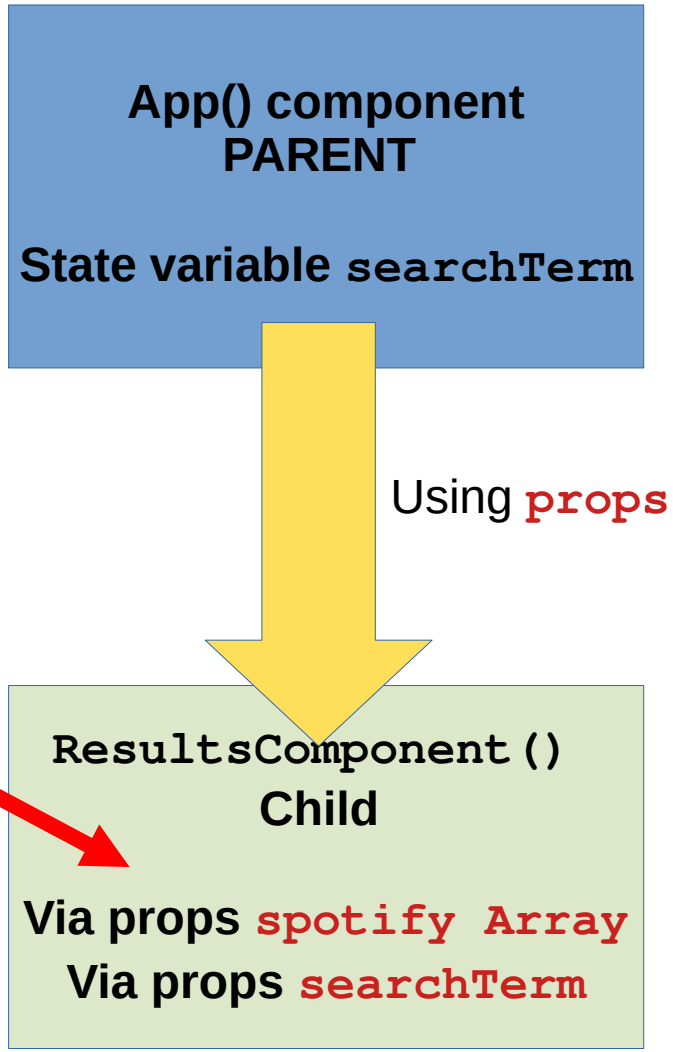
Part 2 – using the child component to render or display results

- 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  }
```



- 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

```javascript
// This is the child component. It is used to display the results
// of the searches conducted. The parent must provide the searchTerm
// and access to the array of JSON objects.

function ResultsComponent(props) {
  // Within this component or function we create our
  // filter function. It will be needed to conduct
  // the search of the array of JSON objects.

  function spotifyFilterFunction(searchTerm) {
    return function (spotifyObject) {
      // convert everything to lower case for string matching
      let artist = spotifyObject.artist.toLowerCase();
      let track = spotifyObject.track.toLowerCase();
      return (
        searchTerm !== "" &&
        (track.includes(searchTerm.toLowerCase()) ||
          artist.includes(searchTerm.toLowerCase()))
      );
    };
  }
```

```json
    {
      "track": "Smells Like Teen Spirit - Remastered 2021",
      "artist": "Nirvana",
      "release": 1991,
      "streams": 1690192927
    },
    {
      "track": "Without Me",
      "artist": "Eminem",
      "release": 2002,
      "streams": 1687664027
    },
    {
      "track": "When I Was Your Man",
      "artist": "Bruno Mars",
      "release": 2012,
      "streams": 1661187319
    },
```

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 resul
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
99            </p>
```

Part 4 – conditional rendering

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

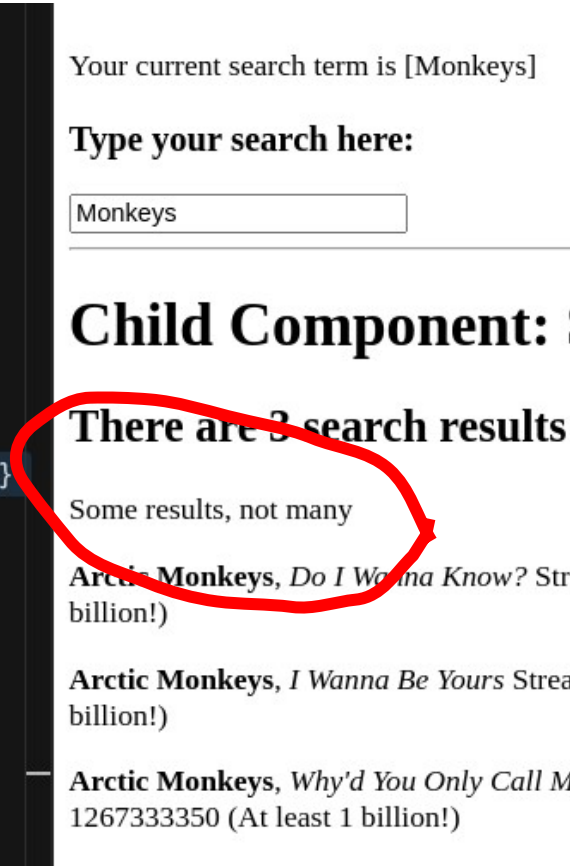
```
{ condition && <div>Write something</div> }

{ !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>
```

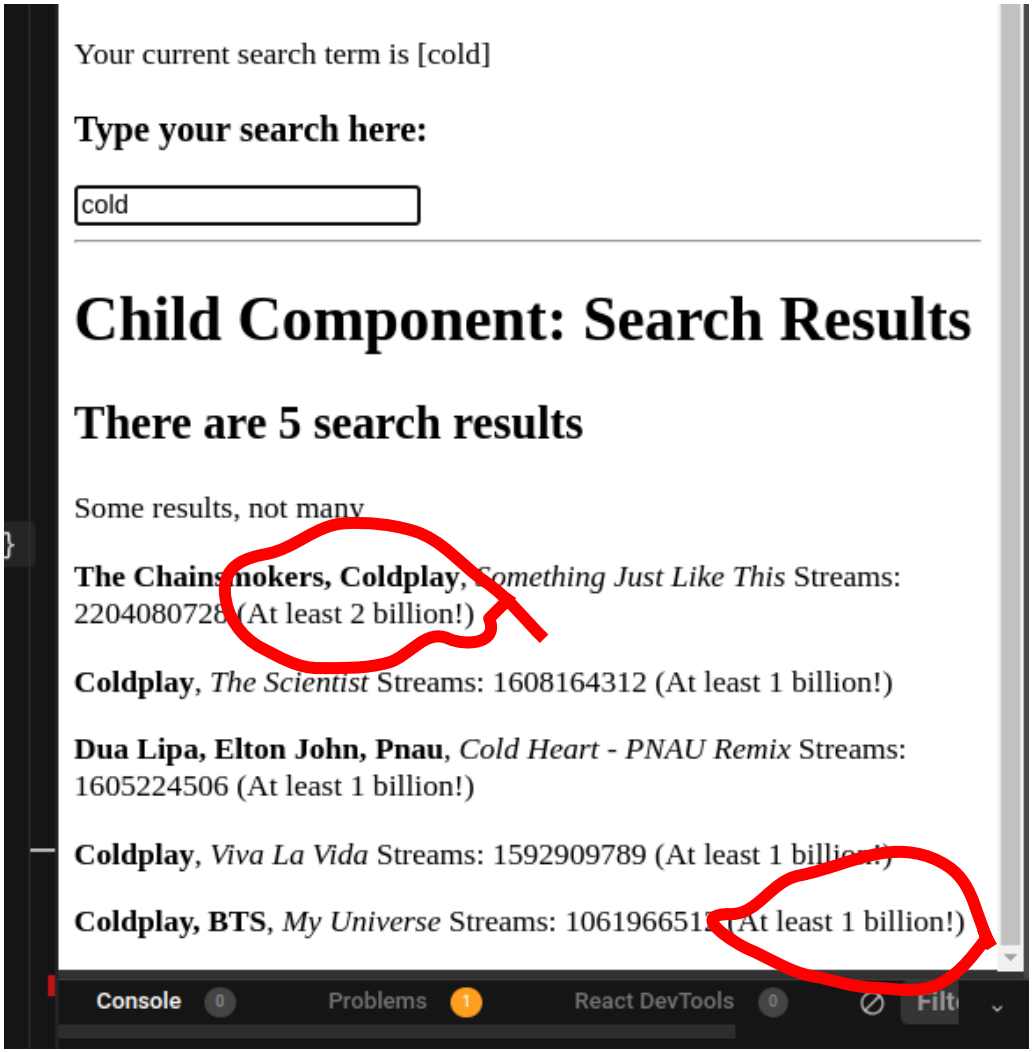


Part 5 – writing our own helper functions for specific tasks

- 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)

```
// Writing our own functions for use in our app
// Writing a comment about a specific number.
function writeNumberAsWords(n) {
  let nAsStr = n.toString(10); // our number is in base 10
  let digits = nAsStr.length; // number of digits
  let firstDigit = nAsStr.charAt(0);

  let message = "nothing";
  if (digits === 9) {
    message = "(At least " + firstDigit + " hundred million!)";
  } else if (digits === 10) {
    message = "(At least " + firstDigit + " billion!)";
  } else {
    message = "(Lots of streams)";
  }

  return message;
}
```

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

```
// Writing our own functions for use in our app
// Writing a comment about a specific number.
function writeNumberAsWords(n) {
  let nAsStr = n.toString(10); // our number is in base 10
  let digits = nAsStr.length; // number of digits
  let firstDigit = nAsStr.charAt(0);

  let message = "nothing";
  if (digits === 9) {
    message = "(At least " + firstDigit + " hundred million!)";
  } else if (digits === 10) {
    message = "(At least " + firstDigit + " billion!)";
  } else {
    message = "(Lots of streams)";
  }

  return message;
}
```

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