08 TimeLine and using fetch API

<https://www.youtube.com/watch?v=c849NW2qnS8//>

The breaking bad api

App.js

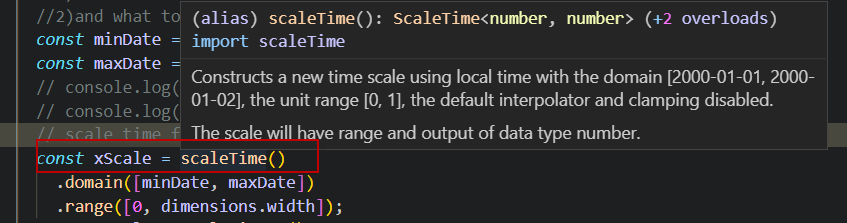
Making calls to 2 endpoints in 2 use effect hooks and saving the data from the calls in 2 useState hooks.

Using map to make character selection list. Once selected passing it and episodes to the BBTimeline component.

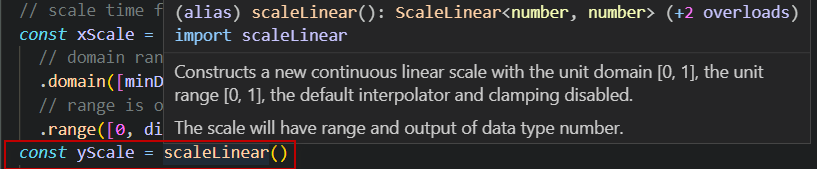


In BBTimeline

scaleTime() – D3 import that maps dates to the available space in the SVG. Also position episode on x axis depending on the time the episode aired. Map dates to pixel values on x axis



scaleLinear()



Commented App.js

import React, { useState, useEffect } from "react";

import BBTimeline from "./BBTimeline";

import "./App.css";

*function* App() {

  // bbEpisodes is the value of the the data key

*const* [bbEpisodes, setBbEpisodes] = useState([]);

  //bbCharacters makes the drop down selection

*const* [bbCharacters, setBbCharacters] = useState([]);

  //  getting highlight from the onChange event that calls

  // the setHighlight method

*const* [highlight, setHighlight] = useState();

  useEffect(() *=>* {

    fetch("https://www.breakingbadapi.com/api/characters?category=Breaking+Bad")

      .then((*response*) *=>* response.ok && response.json())

      .then((*characters*) *=>* {

        // useState method for getting the array of

        // characters passed into state

        // use to make drop down selection

        setBbCharacters(

          characters.sort((*a*, *b*) *=>* a.name.localeCompare(b.name))

        );

      })

      .catch(console.error);

  }, []);

  useEffect(() *=>* {

    fetch("https://www.breakingbadapi.com/api/episodes?series=Breaking+Bad")

      .then((*response*) *=>* response.ok && response.json())

      .then((*episodes*) *=>* {

        console.warn(episodes);

        // useState method to get episodes array passed into

        // state with key of data

        setBbEpisodes(episodes);

      })

      .catch(console.error);

  }, []);

  return (

    <React.Fragment>

      <h1>Breaking Bad Timeline</h1>

      <BBTimeline highlight={highlight} data={bbEpisodes} />

      <h2>Select your character</h2>

      <select value={highlight} onChange={(*e*) *=>* setHighlight(e.target.value)}>

        <option>Select character</option>

        {bbCharacters.map((*character*) *=>* (

          <option key={character.name}>{character.name}</option>

        ))}

      </select>

    </React.Fragment>

  );

}

export default App;

commented BBtimeline.js

//https://www.youtube.com/watch?v=c849NW2qnS8//

import React, { useRef, useEffect } from "react";

import { select, min, max, scaleTime, scaleLinear, axisBottom } from "d3";

import useRsizeObserver from "./useResizeObserver";

// parsing the date strings from the API in this helper method

// turns the string into a real date

*const* getDate = (*dateString*) *=>* {

*const* date = dateString.split("-");

  return new Date(date[2], date[0] - 1, date[1]);

};

// getting data and highlight from app.js

*function* BBTimeline({ *data*, *highlight* }) {

*const* svgRef = useRef();

*const* wrapperRef = useRef();

  // getting dimensions of svg

*const* dimensions = useRsizeObserver(wrapperRef);

  // useEffect to access the svg element to work with d3

  useEffect(() *=>* {

*const* svg = select(svgRef.current);

    //first time through no dimensions

    if (!dimensions) return;

    // figure out when first and last episode aired to get min and max

    //for the xScale

    //1) tell d3 where to look for data

    //2)and what to compare inside the data array

*const* minDate = min(data, (*episode*) *=>* getDate(episode.air\_date));

*const* maxDate = max(data, (*episode*) *=>* getDate(episode.air\_date));

    // console.log(minDate);

    // console.log(maxDate);

    // scale time from D3 helps map dates to pixel valies on x axis

*const* xScale = scaleTime()

      // domain range of input values

      .domain([minDate, maxDate])

      // range is ouput value 0 to the total width of svg

      .range([0, dimensions.width]);

    // yScale use scaleLinear()- height will depend on the number of main

    //characters in that episode.  Map the number of main characters

    // to the amount of space in the SVG

*const* yScale = scaleLinear()

      // input the max amount of characters in the  data array min is 0

      // tell max how to compare the episodes to find the max value

      .domain([max(data, (*episode*) *=>* episode.characters.length), 0])

      //output - the pixel values range 0 to height of svg

      .range([0, dimensions.height]);

    /////////// need a line for each episode

    svg

      //d3 select all the existing elements with class of episode

      .selectAll(".episode")

      // sync with this data

      .data(data)

      // for every episode that is still missing create a new

      //line element and attach class episode to them

      // so that they update later

      .join("line")

      .attr("class", "episode")

      // define the coordinates of each line and color the line

      // lines have 4 attr for the xy of start

      //and the xy of end of line

      // define a call back function for episode, if episode.characters

      // includes highlight from the character selector drop down in app.js

      // color it blue ,else black

      .attr("stroke", (*episode*) *=>*

        episode.characters.includes(highlight) ? "blue" : "black"

      )

      // the position of the start on the x axis depends on airing date

      // pass in call back that recives the current episode in array as

      // an argument  get pixel value of the date from xScale

      .attr("x1", (*episode*) *=>* xScale(getDate(episode.air\_date)))

      // this is very bottom of the svg

      .attr("y1", dimensions.height)

      // x2 is the same as x1 since this is a vertical line

      .attr("x2", (*episode*) *=>* xScale(getDate(episode.air\_date)))

      // the height of the line - pass in callback with the current episode

      // pass the characters.length to the yScale

      .attr("y2", (*episode*) *=>* yScale(episode.characters.length));

    // axisBottom takes in the xScale defined above

    // axis bottom refers to where the date in the scale label arre appearing,

    // not the axis position on the chart, will need transform, translate

    // for that.

*const* xAxis = axisBottom(xScale);

    // tell d3 to position the xAxis

    svg

      // put in the g element with className of x-axis

      .select(".x-axis")

      // call xAxis - renders the x axis that d3 is generating

      // inside the g element

      .call(xAxis)

      // this puts the axixs at bottom of chart

      //moves the axis to the bottom of the chart

      .style("transform", `translateY(${dimensions.height}px)`)

      .call(xAxis);

  }, [data, dimensions, highlight]);

  return (

    <div ref={wrapperRef} style={{ marginBottom: "2rem" }}>

      <svg ref={svgRef}>

        {/\* creating this x-axis g element to render the x axis into into \*/}

        <g className="x-axis"></g>

      </svg>

    </div>

  );

}

export default BBTimeline;