01 Basics of combining React and D3 using Hooks and the D3 Join API

<https://www.youtube.com/watch?v=Y-ThTzB-Zjk>

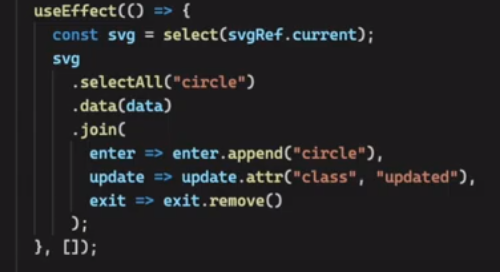
use create-react-app for set up

Let D3 handle the DOM

Let React provide the SVG to D3 using the useRef hook

D3 General Update Pattern –

12:54 into second video



Commented code from the video

// https://www.youtube.com/watch?v=9uEmNgHzPhQ&t=257s// the original

//https://www.youtube.com/watch?v=Y-ThTzB-Zjk // re recoreded

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

import { select } from "d3";

import "./App.css";

*function* App() {

*const* [data, setData] = useState([25, 30, 45, 60, 28]);

  // this provides creates the referance object- will get passed to the DOM

*const* svgRef = useRef();

  // console.log(svgRef); //this console.log is undefined - have to wait one cycle to access the svg dom element

  // the useEffect hook is the best place to access the svg DOM element

  useEffect(() *=>* {

    // the call back in the useEffect function gets called for the first time when the DOM elements have been rendered

    // and rendered when the elements in the dependancy array (at bottom ) change

    // console.log(svgRef);

    //

    // making use of D3 select to make the SVG avalible to D3

*const* svg = select(svgRef.current);

    // now have all the d3 methods avalible

    // console.log(svg.selectAll("circle").data(data)); // update enter and exit are here

    svg

      // select all the circle elements in the svg

      .selectAll("circle")

      // syncronize the selected circles with the data array 9:44

      .data(data)

      // can now control entering extiting and updating in the join api

      // pass in a call back for each type of element for entering updating exiting

      // this creates the 5 circes (5 elements in the data array)

      // the Join api

      // now can handle all the updateing, entering and exiting here in the join api

      ///////////////////////////////////////////////// the non DRY way

      // one way is to pass in a call back for each - this is not very DRY

      .join(

        // 11:27

        // pass in a call back for each type of element  - entering, updating and exiting

        // enter - every new piece of data that needs to be represented in the SVG

        (*enter*) *=>*

          enter

            .append("circle") // this creates the new circle for each element in the data array

            // for every entering element do this stuff

            .attr("r", (*value*) *=>* value)

            .attr("cx", (*value*) *=>* value \* 2)

            .attr("cy", (*value*) *=>* value \* 2)

            .attr("stroke", "red"),

        // update  - called for every circle already in the svg

        (*update*) *=>*

          update

            .attr("class", "updated") // give each one a class of updated

            // to share the attributes added in enter to any circles already in the SVG use updated add them to the update call back

            .attr("r", (*value*) *=>* value)

            .attr("cx", (*value*) *=>* value \* 2)

            .attr("cy", (*value*) *=>* value \* 2)

            .attr("stroke", "red"),

        // exit - every circle no longer needed - remove the ones not needed

        (*exit*) *=>* exit.remove()

      );

      ////////////////////// The betterway to use join/////////////////////////////////////

      // just add them after the join - join handles entering and updating

      .join(

        // 11:27

        // pass in a call back for each type of element  - entering, updating and exiting

        // enter - every new piece of data that needs to be represented in the SVG

        (*enter*) *=>* enter.append("circle"),

        // update  - called for every circle already in the svg

        (*update*) *=>* update.attr("class", "updated"), // give each one a class of updated

        // exit - every circle no longer needed - remove the ones not needed

        (*exit*) *=>* exit.remove()

      )

      // put attributes go here after the join- this will apply them to both entering and updating

      // the join api returns a selection of both entering and updating elements

      .attr("r", (*value*) *=>* value)

      .attr("cx", (*value*) *=>* value \* 2)

      .attr("cy", (*value*) *=>* value \* 2)

      .attr("stroke", "red");

      ///////////////////////////////////////////////////////////////////////////////////

      // ////////////////////////////////the best way to use the join api//////////////////////////

      // the exit call back is a d3 default call back

      //  (exit) => exit.remove()

      // d3 will automaticaly remove all of the circles that are no longer needed

      // even without explicitly calling it

      // you would want to use the call back if you needed to

      //transition or animate while exiting

      // since we do not need to add a class for this vis can also remove the update callback

      // becasue the enter callback is just appending a circle, can replace the

      //function   (enter) => enter.append("circle"),

      // with just adding string "circle in the join "

      .join("circle")

      .attr("r", (*value*) *=>* value)

      .attr("cx", (*value*) *=>* value \* 2)

      .attr("cy", (*value*) *=>* value \* 2)

      .attr("stroke", "red");

    // elements in this dependancy array will cause the DOM element to re render if they are changed

  }, [data]);

  return (

    <React.Fragment>

      <svg ref={svgRef}></svg>

      <br></br>

      <button onClick={() *=>* setData(data.map((*value*) *=>* value + 5))}>

        Update data by 5

      </button>

      <button onClick={() *=>* setData(data.filter((*value*) *=>* value <= 35))}>

        Filter out more than 35

      </button>

    </React.Fragment>

  );

}

export default App;