06 Responsive resizeObserver and move out of App js

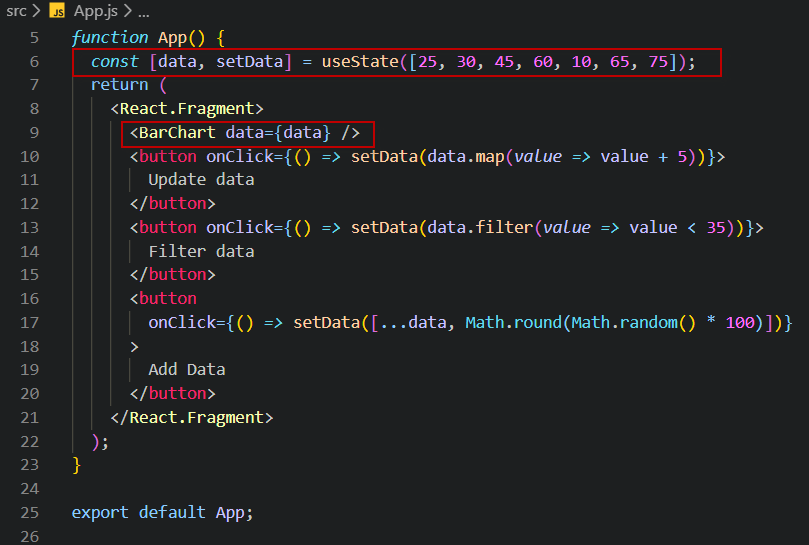
18 min in is summary of the video

<https://www.youtube.com/watch?v=a4rstx9Pz2o>

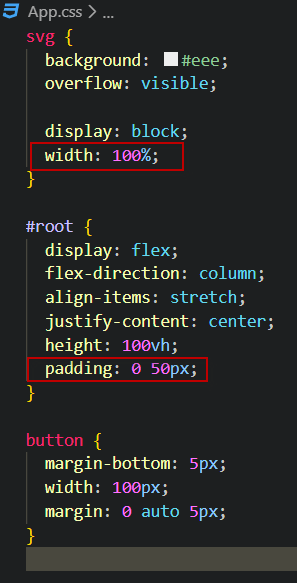
Making the chart responsive using ResizeObserver API in a custom react hook – make the dimensions of the SVG responsive

Extract code for BarChart into its own component to make it reuseable and responsive from there

When moving to the BarChart component leave the data in App js- pass to BarChart in data prop useState



To prepare for responsiveness change the SVG width to 100% and add some padding to root

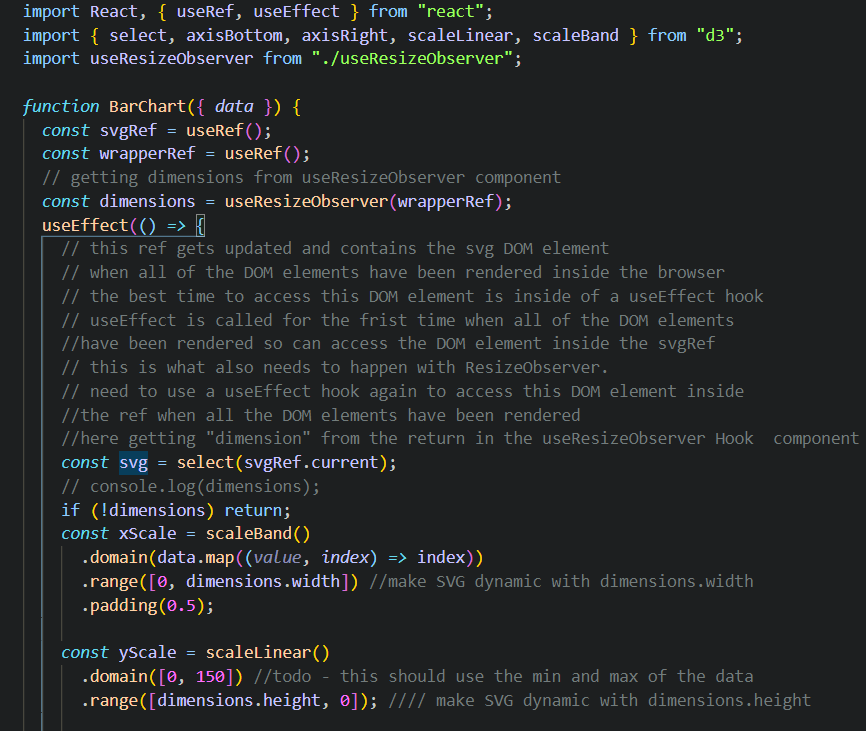


To make the SVG dynamic and responsive need to know the height and width of the SVG at the moment. Its browser. Need to know when the dimensions change too.

Using resize observer API in a custom hook. Will observe the DOM element and return the height and width.

Resizeobserver from resize-obserrver-polyfill allows it to work in older browsers





The commented code from the useResizeObserver.js file

import { useEffect, useState } from "react";

import ResizeObserver from "resize-observer-polyfill";

//the useResizeObserver function recives a reference to an element

*const* useResizeObserver = (*ref*) *=>* {

  // initially there are no dimensions in state.

  //setDimensions will be used by ResizeObserver when the ref being

  //observed changes.

*const* [dimensions, setDimensions] = useState(null);

  // ResizeObserver needs a DOM element to observe.

  //we are using a useRef hook to observe to define our const svgRef

  // svgRef is passed it the return to the svg

  useEffect(() *=>* {

    // need to define what is to be observed

    // console.log(ref.current);

*const* observeTarget = ref.current;

    // define the resizeObserver as a new

    // in the callback pass entries which are being observed

    // and resized as an argument. Entries is an array

*const* resizeObserver = new ResizeObserver((*entries*) *=>* {

      // console.log(entries);

      entries.forEach((*entry*) *=>* {

        // for each entry call the setDimensions function

        // pass the entire contentRect object to

        // store them in the use state hook

        setDimensions(entry.contentRect);

      });

    });

    // telling resizeObserver what to observe

    resizeObserver.observe(observeTarget);

    return () *=>* {

      // this is a clean up function

      // will be called when the component using

      // the resizeObserve hook is getting removed

      // or un mounted

      resizeObserver.unobserve(observeTarget);

    };

    // ref is a dependancy to the use effect hook

  }, [ref]);

  // returns the width and height that have been set by the ResizeObserver API and the useState hook

  return dimensions;

};

export default useResizeObserver;

The commented code from the BarChart component

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

import { select, axisBottom, axisRight, scaleLinear, scaleBand } from "d3";

import useResizeObserver from "./useResizeObserver";

*function* BarChart({ *data* }) {

*const* svgRef = useRef();

  // console.log(svgRef);

  // use the wrapperRef to pass into dimensions from resize observer

*const* wrapperRef = useRef();

  // console.log(wrapperRef);

  // getting dimensions from useResizeObserver component

*const* dimensions = useResizeObserver(wrapperRef);

  // console.log(dimensions);

  useEffect(() *=>* {

    // console.log(svgRef.current);

    // this ref gets updated and contains the svg DOM element

    // when all of the DOM elements have been rendered inside the browser

    // the best time to access this DOM element is inside of a useEffect hook

    // useEffect is called for the frist time when all of the DOM elements

    //have been rendered so can access the DOM element inside the svgRef

    // this is what also needs to happen with ResizeObserver.

    // need to use a useEffect hook again to access this DOM element inside

    //the ref when all the DOM elements have been rendered

    //here getting "dimension" from the return in the useResizeObserver Hook  component

*const* svg = select(svgRef.current);

    // console.log(dimensions);

    // the first trip through dimensions are null, so in that case just return

    if (!dimensions) return;

*const* xScale = scaleBand()

      .domain(data.map((*value*, *index*) *=>* index))

      .range([0, dimensions.width]) //make SVG dynamic with dimensions.width

      .padding(0.5);

*const* yScale = scaleLinear()

      .domain([0, 150]) //todo - this should use the min and max of the data

      .range([dimensions.height, 0]); //// make SVG dynamic with dimensions.height

*const* colorScale = scaleLinear()

      .domain([75, 100, 150])

      .range(["green", "orange", "red"])

      .clamp(true);

*const* xAxis = axisBottom(xScale).ticks(data.length);

    svg

      .select(".x-axis")

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

      .call(xAxis);

*const* yAxis = axisRight(yScale);

    svg

      .select(".y-axis")

      .style("transform", `translateX(${dimensions.width}px)`)

      .call(yAxis);

    svg

      .selectAll(".bar")

      .data(data)

      .join("rect")

      .attr("class", "bar")

      .style("transform", "scale(1, -1)")

      .attr("x", (*value*, *index*) *=>* xScale(index))

      .attr("y", -dimensions.height)

      .attr("width", xScale.bandwidth())

      .on("mouseenter", (*value*, *index*) *=>* {

        svg

          .selectAll(".tooltip")

          .data([value])

          .join((*enter*) *=>* enter.append("text").attr("y", yScale(value) - 4))

          .attr("class", "tooltip")

          .text(value)

          .attr("x", xScale(index) + xScale.bandwidth() / 2)

          .attr("text-anchor", "middle")

          .transition()

          .attr("y", yScale(value) - 8)

          .attr("opacity", 1);

      })

      .on("mouseleave", () *=>* svg.select(".tooltip").remove())

      .transition()

      .attr("fill", colorScale)

      .attr("height", (*value*) *=>* dimensions.height - yScale(value));

    // dimensions is a dependancy for this useEffect hook

  }, [data, dimensions]);

  // need to wrap the SVG in a div, resize observer is not

  // working when when passed to SVG directly

  return (

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

      <svg ref={svgRef}>

        <g className="x-axis"></g>

        <g className="y-axis"></g>

      </svg>

    </div>

  );

}

export default BarChart;