Tuto4: Developing Data Visualizations with React and D3.js

Nicolas Médoc Luxembourg Inst. of Science and Technology

Outline

1. Separation of responsibilities: React component + D3 class
2. React useEffect() hook
3. React useRef() hook
4. Creation of a scatterplot visualization

Combining D3.js and React: Separation of responsibilities

- **D3.js visualizations** are implemented in **self-contained class**, without any dependencies to React library
- For each visualization, **one React component** implements the container of the visualization to makes the glue between the D3js class and the React application.
- The **React component is responsible to:**
  - get the data from the Redux store;
  - instantiate the D3.js class and call render/update methods to build/update the visualization,
  - provide the event handler methods to D3.js class to update the store.

Combining D3.js and React: first example

- `https://github.com/nicolasmedoc/TD4-D3jsInReact.git`
- Matrix-d3.js
  - is a **javascript class** which will renders the visualization. It is self-contained and independent from React
  - declares a method **create()** to initialize the SVG element
  - declares a method **clear()** to remove the SVG from the DOM
  - declares **one or several update methods** (e.g. renderMatrix(), updateCells()) to change the view or a part of the visualization when the data changes, with the **global update pattern**.

Combining D3.js and React: first example

- Matrix.js (React component)
  - is the container of the matrix view. It uses the React state (useState() hook) or the Redux store (useSelector() hook) and dispatch the actions of reducers to handle the events.
  - controls the component lifecycle with **useEffect() hook** when the component **did mount**, **did update** (when data changes) or **did unmount** (when removed from the page)
  - **instantiates the d3 class** and store it in a **Ref** (with useRef() hook) to keep the d3 instance when the component re-render.
  - **renders the ** element containing the SVG and **stores it in a Ref** (with useRef() hook) to avoid re-render it.

React useEffect() hook to control the component life cycle

3 functions are declared in 2 different profiles of the useEffect() hook to declare additional behavior in the React component life cycle:

```
import { useEffect } from 'react';
...
useEffect(()=>{
    // the behavior after the component creation (did mount)
    return ()=>{
        // the behavior after the component deletion (did unmount)
        // is declared in the return function
    }
}, []) // empty array
useEffect(()=>{
    // behavior after update of dependency1 or dependency2 only
},[dependency1,dependency2]) // array of dependency variables
useEffect(()=>{
    // behavior after update of dependency3 only
},[dependency3]) // array of dependency variables
```

React useRef() hook to persist a value in the component

Used to persist the instance object of the D3 class in a React component:

```
import { useEffect, useRef } from 'react';
...
const divContainerRef=useRef(null);
const matrixD3Ref = useRef(null)
vuseEffect(()=>{
    const matrixD3 = new MatrixD3(divContainerRef.current);
    matrixD3Ref.current = matrixD3;
},[])
```

React useRef() hook to persist a value in the component

Used to persist the previous value in the sub part of a Redux slice:

```
import { useEffect, useRef } from 'react';
...
const dataSliceAttributeRef = useRef(null)
useEffect(()=>{
    // behavior after update of dataSlice
    if(dataSliceAttributeRef.current!==dataSlice.attribute){
        // attribute has been updated => do something
        // e.g. call specific update method in D3 class
        dataSliceAttributeRef.current = dataSlice.attribute
    }
},[dataSlice]) // array of dependency variables
```

Building a scatterplot: getting the data set from Redux store

in components/scattterplot/ScatterplotContainer.js, get the matrixData slice from the redux store with useSelector() hook

```
import {useSelector} from 'react-redux'
...
function ScatterplotContainer(){
    const matrixData = useSelector(state =>state.matrix)
...
```

And call the scatterplotD3.renderScatterplot() method in the useEffect hook handling matrixData updates:

```
scatterplotD3.renderScatterplot(matrixData.genData,xAttribute,yAttribute,
controllerMethods);
// controllerMethods being already declared
// with empty handleOnClick, handleOnMouseEnter and handleOnMouseLeave
```

Building a scatterplot: creation of scales and X/Y axis

in components/scattterplot/Scatterplot-d3.js, in the method updateAxis():

Exercice1: using d3.min(valueAccessor) and d3.max(valueAccessor), set the domain values of this.xScale.domain(...) and this.yScale.domain(...) to put **nbProductSold in X Axis and salesGrowth in Y axis**.

And use these scales to build X axis and Y axis (.xAxisG and .yAxisG are built in create() function):

```
this.matSvg.select(".xAxisG")
    .transition().duration(500)
    .call(d3.axisBottom(this.xScale))
;
this.matSvg.select(".yAxisG")
    .transition().duration(500)
    .call(d3.axisLeft(this.yScale))
```

Building a scatterplot: update dot positions with scales

in components/scattterplot/Scatterplot-d3.js, in the method updateDots():
Exercice2: using X/Y scales, apply a translation to .dotG to update the dot positions. updateDots(selection) is called from renderScatterplot() with in parameter the selection of .dotG, built with the update pattern.

```
selection
    .transition().duration(500)
    .attr("transform", (itemData)=>{
        // use scales to return shape position from data values
})
```

Building a scatterplot: interactions

Exercice 3: in the useEffect() hook in ScatterplotContainer.js, call dispatch to trigger the reducers of matrixData and matrixSync (mouse hover and click interactions).

Observe the sequence of logs when clicking an item in the matrix view or the scatterplot:

- the Redux store is update in the matrixData slice
- the two components re-render because they call useSelector() on matrixData
- the useEffect with matrixData dependencies are called from Matrix.js and ScatterplotContainers.js
- the update pattern call updateDots() in renderScatterplot() of Scatterplot-d3.js and call updateCells() in renderMatrix() of matrix-d3.js