**PROYECTO CONTADOR**

Los proyectos en React suelen estar modularizados, esto significa que se crean varios archivos para concretar su funcionalidad.

Cuando se usa babel y se quiere modificar el **type** de los scripts en un archivo .js externo, basta con agregar el atributo type=”text/babel” para que este funcione.  
Esto es debido a que el atributo type no solo se usa cuando se introducen los scripts en el HTML, sino también en estos casos.

**INICIANDO**

Para crear esta aplicación se van a requerir fundamentalmente:

* Un index.html
* Un index.js
* Un Contador.js (importante la nomenclatura así)

En el index.html, además de incluir el cdn de react y babel, se crea un div#root en el cual se renderizan los resultados.

En el index.js, solo se invoca el renderizado del elemento creado en **Contador.js**

ReactDOM.render(

<Contador></Contador>, document.querySelector('#root')

);

Este es todo el código del index.js

En el **Contador.js** se crea todo el elemento. Es importante incluir en el index.html primero 🡪 Contador.js y luego 🡪 index.js

**CODIGO DEL CONTADOR.JS**

Para este se crea una const Contador que como arrow function retorna contenido html.

Además, se crea un estado para manipularle y que el objeto se renderice cada vez que este **state** se modifique.

const Contador **=** () => {

    const [state, setState] **=** React.useState(0);

    const aumentar**=** () => {

        if(state **<=** 0){

            setState(state **+** 1);

        } else {

            setState(state **+** state);

        }

    }

    const disminuir **=** ()=> {

        setState(state **-** 1);

    }

    const restablecer **=** () => {

        setState(0);

    };

    return (

    <div>

        <h2 *className***=**{ state **>=** 0 **?** "mayor" **:** "menor"} *id***=**"title">Contador: {state}</h2>

        <hr></hr>

        <button *id***=**"btnAumentar" *onClick***=**{aumentar}>Incrementar</button>

        <button *id***=**"btnDisminuir" *onClick***=**{disminuir}>Disminuir</button>

        <button *id***=**"btnRestablecer" *onClick***=**{restablecer}>Restablecer</button>

    </div>

    )

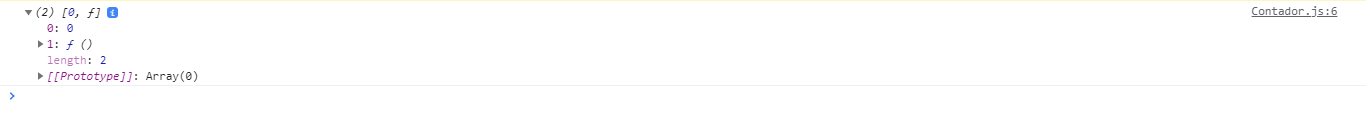
};

Este es todo el código del Contador.js

En la definición de una Constante para ser renderizada, puedo declarar:

const state **=** React.useState(0);

Imprimiendo esto en consola, podemos ver que se trata de un array, que el primero es un valor y el segundo una funcion



Ahí se entiende que los **state** se traten de esta forma

  const [state, setState] **=** React.useState(0);

**RETURN**

Lo más importante luego de la declaración del estado es el return. Este está compuesto por un div, que dentro tiene un <h2>, <hr> y 3 <button>

* El h2 es el que se encarga de mostrar el valor del **state**, por eso se incluye un {} template al **state** declarado de React.useState();
* Los botones, uno para aumentar, disminuir y restablecer, son los que se encargan de manipular el state y hacer que el elemento se renderice.  
  Como se puede ver estos botones tienen agregado un atributo **onClick** que es el equivalente en HTML a agregar un eventListener en JS. En este se usa un {} para invocar una función.  
  Estas funciones están anteriormente definidas. Definidas arrow functions como **const** en las cuales se invoca a setState para modificar su valor.
* Se agregó una funcionalidad que es determinar el color del h2, dependiendo del valor de state. Si es >0 verde, si no, rojo. Esto se realiza con el operador ternario dentro del atributo className del elemento h2  
    
  className={ state >= 0 ? "mayor" : "menor"}  
    
  como se ve, se usan {} ya que es código JS. No se usa el atributo class simplemente porque resulta ambiguo con la palabra reservada **class** usada para definir clases POO.

**RENDERIZANDO**

Por ultimo, se implementa la renderización de la “nueva etiqueta” en el index.js

ReactDOM.render(

<Contador></Contador>, document.querySelector('#root')

);