**Custom hooks**

Los customs hooks pueden utilizarse para “limpiar” las declaraciones e importaciones de un componente y “delegar” o “modularizar” las mismas.

Por ejemplo, podríamos considerar un componente Custom.jsx en el cual contengamos un contador (state) y algunos botones que modifiquen el mismo. Una forma de customizar un hook es de la siguiente manera:

Este es el código para el componente:

import { useCustom } from "../hooks/useCustom";

const Custom = () => {

  const [contador, increment, decrement] = useCustom(-64, 8);

  return (

    <div>

      <p>Este es el custom hook : {contador}</p>

      <button *onClick*={decrement}>-</button>

      <button *onClick*={increment}>+</button>

    </div>

  );

};

export default Custom;

notese que se desestructura el retorn de useCustom para utilizar **contador, increment, decrement.**

y en un nuevo archivo useCustom.js tendríamos:

import { useState } from "react";

export const useCustom = (*initialValue*, *range* = 1) => {

    const [contador, setContador] = useState(*initialValue*);

    const increment = () => {

        setContador(contador + *range*);

    };

    const decrement = () => {

        setContador(contador - *range*);

    };

    return [contador, increment, decrement]

}

En este caso, se personaliza useCustom para que reciba el valor inicial y el “rango” de salto del contador. De dicha forma es más reutilizable el hook.

Esto mismo puede realizarse con cualquier componente para “Limpiar” el código y solo renderizar las etiquetas necesarias.

**HACIENDO UN CUSTOM HOOK DEL COMPONENTE MEMO.JSX**

Una buena forma de limpiar el componente sería creando un custom hook para el mismo. En este caso, se crea el archivo useMemorize.js (notese que es js y no jsx)

Desde este se manejan las importaciones de react y este es su código:

import { useState, useMemo, useCallback } from 'react'

export const useMemorize = () => {

*//estado contador*

    const [counter, setCounter] = useState(0);

*//estado independiente*

    const [fondo, setFondo] = useState(true);

*//funcion para alterar state independiente*

    const handleFondo = () => {

        fondo

            ? (document.querySelector("body").style.backgroundColor = "lightgreen")

            : (document.querySelector("body").style.backgroundColor = "crimson");

    };

*//Funcion para explicar useMemo*

    const procesar = (limite) => {

        for (let i = 0; i < limite; i++) {

            console.log(`numero: ${i}`);

        }

        return `finalizó el proceso luego de ${limite} veces`;

    };

*//usando useMemo*

    const memorizar = useMemo(() => procesar(counter), [counter]);

*//usecallback*

    const add = useCallback(() => {

        setCounter((actual\_value) => actual\_value + 1);

    }, [setCounter]);

    return [counter, fondo, memorizar, add, handleFondo, setFondo];

}

Ahora, desde aquí se manejan todas las operaciones, pero en el retorno no se incluyen algunas funciones o variables que se quiere que no se manipulen desde el componente. Por ejemplo, la funcion procesar no se retorna, en su lugar se retorna o deja disponible el use memo **memorizar** que espera por la dependencia **counter.**

Lo mismo ocurre con setCounter, que no se retorna directamente, sino que se retorna una funcion que le manipule, restringiendo el acceso al mismo desde el componente Memo.jsx o el que vaya a importar el archivo **useMemorize.js**

Del lado del componente, este es el código

import { useMemorize } from "../../hooks/useMemorize";

const Memo = () => {

  const [counter, fondo, memorizar, add, handleFondo, setFondo] = useMemorize();

  return (

    <>

      {handleFondo()}

      <h2 className="bg-warning p-3">utilizando memo y UseMemo</h2>

      <p>

        Counter: <Dato value={counter}></Dato>

      </p>

      <br />

      <Callback funcion={add}></Callback>

      <br />

      <br />

      <br />

      <button

        onClick={() => {

          handleFondo();

          setFondo(!fondo);

        }}

      >

        State independiente <br />

        (backgroundColor)

      </button>

    </>

  );

};

export default Memo;

lo que es necesario realizar es una destructuracion del array retornado:

  const [counter, fondo, memorizar, add, handleFondo, setFondo] = useMemorize();

y ya es posible utilizar las funciones y constantes provenientes del archivo useMemorize.js