**React.Fragment**

podemos usarlo para declarar mas de un boton a la hora de hacer un root.render

root.render(

<React.Fragment>

<button>Hola button</button>

<button>Hola button</button>

</React.Fragment>

)

**se puede sustituos por**

root.render(

<>

<button>Hola button</button>

<button>Hola button</button>

</>

)

**Tienen que ser declarativos**

Lo que pongamos tiene que ser quiro que salga esto no creame un boton

bien

root.render(

<React.Fragment>

<Button text='button 1' />

<Button text='button 2' />

<Button text='button 3' />

</React.Fragment>

)

mal

root.render(

<React.Fragment>

{createButton({text:'button1'})}

{createButton({text:'button2'})}

{createButton({text:'button3'})}

</React.Fragment>

)

funcion

const Button = ({text}) => {

return <button>{text}</button>

}

componente elemento diferencia

componente es la funcion que devuelve un elemento

y elemneto es el bloque en si

TwitterFollowCard

username='manu16'>

Manuel Muñoz Felix

</TwitterFollowCard>

**Children en una funcion**

Cuando en un componente pone como variable de entrada childen representara a todo lo que envuelve

**Hooks**

**useState from 'react'**

import { useState } from "react"

const state = useState(false) // le indicamos el estado por defecto

//Se crea un array donde el primer valor del array sera el estado y el segundo el interruptor para cambiarlo

const isFollowing = state[0]

const setIsFollowing = state[1]

// podemos desestructurar

const [isFollowing, setIsFollowing] = useState(false)

es asincrono si pongo un log despues de actualizarlo no lo tendrá en cuenta

el useState tiene que estar en el cuerpo de el componente no puede estar en un if ya que hay un array interno que para renderizar va ejecutando los useState por la posicion el la que estan y si hay un if pierde la posicion

**Use Effect**

nos permite ejecutar codigo albratirio cuando el componente se monta en el doom y cada vez que las dependencias que nosotros le digamos cada vez que cambien ejecuta el codigo

useEffect(funcion, [...dependencias])

dependencias == variables cuando cambies estas variables se ejecutara la funcion

para dejar de usar un efecto no vastara con no llamarlo, hay que limpiarlo

// metodo para limpiar el efecto

return () => {

window.removeEventListener('pointermove', handleMove)

}

con getEventsListener(window) puedo mirar los eventos para ver cuando estamos limpiando y que eventos tenemos

**Configurar linter monorepositorio**

- buscar standard js

ejecutamos en el proyecto

npm install standard -D

-D como entorno de desarrollo

añadir en el package.json

"eslintConfig": {

"extends": "./node\_modules/standard/eslintrc.json"

}

crear un file .gitignore

dentro meter:

package-lock.json

node\_modules

**Crear proyecto**

npm create vite@latest

lo creamos en react javascript SK...

podemos nombre

cd 03-mouse-follower/

npm install

npm run dev

**Crear proyecto desde 0 Importante saber esto para las pruebas tecnicas**

npm create vite@latest

- instalamos los plugins

npm install @vitejs/plugin-react -E

- abrimos el visual en el repositorio

code .

- instalamos react

npm install react react-dom -E

- configuramos el vite

creamos file vite.config.js y metemos

import { defineConfig } from "vite";

import react from "@vitejs/plugin-react";

export default defineConfig({

plugins: [react()]

})

- nos vamos a main.js

cambiamos el nombre de main.js a main.jsx

importamos

import { createRoot } from 'react-dom/client'

-cremos el root que apunta al index.html y dento a app

const root = createRoot(document.getElementById('app'))

root.render(<h1>Hello, World!</h1>)

en index.html

cambiamos en el <script> el src="/main.jsx"

Ahora **instalamos linter**

por ultimo creamos una carpeta src y creamos dentro el app.jsx

**Custom hooks**

es reutilizar logina de nuestros componentes en otros componentes

hay que poner use como prefijo en una funcion con hooks

**Hacer test basico**