Desestructuración

Desestructuración de objetos

```
const Persona = {
    nombre:'Tony',
    edad: 40,
    clave: 'Ironman'
}

const retornaPersona = (usuario) =>{
    const {nombre, edad, clave} = usuario
}

retornaPersona(Persona)
```

- La forma más usada es desestructurar desde el parámetro de la función, poniendo las propiedades que interesan

```
const retornaPersona = ({nombre, edad}) =>{

}

retornaPersona(Persona)
```

- Puedo añadir valores por defecto en la desestructuración

```
const retornaPersona = ({nombre, edad, rango="cadete"}) =>{
    console.log(nombre,edad,rango)
}

retornaPersona(Persona)
```

- Si existe esa propiedad en el objeto, usa esa propiedad, si no, la que le definí yo
- No importa el orden en la desestructuración
- Voy a renombrar la función a useContext

```
const useContext = ({edad, clave, nombre, rango =""}){
    return {
        nombreClave: clave,
        anios: edad
    }
}

const avenger = useContext( persona )

console.log(avenger)
```

- Renombrando la función, obtengo el resultado esperado con ambos valores.
    - UseContext es una función como cualquier otra
- Para desestructurar, lo mismo

```
const {nombreClave, anios} = avenger
```

- Si yo necesitara la latitud y longitud se perdería latlng en la desestructuración anidada y quedarían lat y lng como constantes

```
const useContext = ({edad, clave, nombre, rango =""}){
    return {
        nombreClave: clave,
        anios: edad
        latlng:{
            lat: 032,
            lng: 1200
        }
    }
}
const {nombreClave, anios,latlng:{lat, lng}} = useContext(Persona)
```

- Si se quiere conservar latlng se puede hacer así

```
const {nombreClave, anios,latlng} = useContext(Persona)
const {lat, lng} = latlng
```

Desestructuración de arreglos

- En los arreglos la desestructuración se hace por posición. -Cuando no quiero esa posición la marco con una coma

```
const personajes = ['Picasso', 'Monet', 'Schiele']

const [ Pablo, , Egon ] = personajes
```

- Asi puedo trabajar con los valores del return de una función sin parámetros

```
const retornaArreglo =() =>{
    return ['ABC', 123]
}

const [letras, numeros] = retornaArreglo()
```

```
const useState = (valor) =>{
    return [valor, ()=>{console.log("Hola mundi!")}]

}
    const arr = useState('Dalí')

//yo podría llamar a esa función así

arr[1]()    //retorna Hola mundi!
```

- Los parámetros podrían ser nombre y setNombre en la desestructuración

```
const [nombre, setNombre] = useState('Dalí')

console.log(nombre)     //retorna Dalí
setNombre();            //retorna Hola mundi!
```

Promesas

- Las promesas son asincronas por naturaleza, primero se va a ejecutar todo el código sincrono, luego la promesa.
- Trabajan con dos valores llamados por convención *resolve* --> resuelve, y *reject* ---> falla
    - Resolve:
    - Reject:
- Se ejecutan con un callback
- Aquí uso un setTimeout para ver cómo se ejecuta dos segundos -->2000<-- después de recargar el navegador

```
const promesa = new Promise ((resolve, reject) =>{

    setTimeout(()=>{
        resolve();
    }, 2000 )

})
```

3 métodos principales

- .then--> cuando resuelve
- .catch--> cuando da error
- finally--> se ejecuta despues del .then o el .catch
- Aquí puedo apreciar como se imprime en pantalla después de resolver el setTimeout de 2 segundos

```
promesa.then(()=>{

    console.log('Then de la promesa')
})
```

- Con una petición http sería algo así:
    - Resolve envía el resultado al .then

```
const promesa = new Promise ((resolve, reject) =>{
```

```
    const heroe = getHeroeById(2)
    resolve(heroe)

})

promesa.then(()=>{
    console.log('heroe', heroe)
})
```

- Si fuera un error se maneja así con el .catch

```
const promesa = new Promise ((resolve, reject) =>{

    const heroe = getHeroeById(2)
    reject(heroe)

})


promesa.then(()=>{

    console.log('heroe', heroe)

}).catch( err => console.warn( err ))
```

- Realizando un return, la función no retiorna void o undefined, y así puedo pasarle el id

```
const getHeroeByIdAsync = (id)=>{

    return new Promise ((resolve, reject) =>{

        const p1 = getHeroeById(id)
        resolve(heroe)
    })


}
getHeroeByIdAsync(4).then(console.log('heroe', heroe))
```

- Para manejar la excepción, en el caso de error cuando el catch o el then, cuando el primer y unico parámetro es enviado como parámetro de una función, te puedes ahorrar código, y dejar solo el return

```
const getHeroeByIdAsync = (id)=>{

    return new Promise ((resolve, reject) =>{
        if(p1){
            resolve(p1)
```

```
        }else{
            reject('No se pudo encontrar el heroe')
        }

    })


}
getHeroeByIdAsync(4)
//.then(console.log('heroe', heroe))

.then(console.log)
.catch(console.warn)
```

Fetch Api

- Para usar esta API se necesita una apiKey, que será 1234abc
- Necesito el endpoint, lo busco en la API, que será http:endpoint.random?apikey=
- Cojo el endpoint acabado en un signo de interrogacio y copio la apiKey
    - http:endpoint.random?apikey=?1234abc
    - Esto da una respuesta con un JSON lleno de info -Solo inetersa data, imgs >original, imgs > mp4 y similares
- Uso un template literal para hacer la apiKey dinámica

```
const apiKey = '1234abc'

const peticion = fetch(`http:endpoint.random?apikey=${ apiKey}` )

peticion.then(response=>{
    console.log(response)
})
.catch(console.warn)
```

- Obtengo una respuesta en consola, pero no puedo obtener la data, necesito el .json

```
const peticion = fetch(`http:endpoint.random?apikey=${ apiKey}` )

peticion.then(response=>{
    response.json().then( data => {
        console.log(data)
    })
})
.catch(console.warn)
```

- Se puede escribir con menos código, encadenando las promesas
- Ahora si puedo obtener info de la data

```
const peticion = fetch(`http:endpoint.random?apikey=${ apiKey}`;

peticion
    .then (resp => resp.json())
    .then(data =>{
        console.log(data.images.original.url)
    })
    .catch (console.warn)
```

- Puedo usar la desestructuración de la data para no usar data.data, y la url de la ruta, para luego insertar la imagen en el html

```
const peticion = fetch(`http:endpoint.random?apikey=${ apiKey}`;

peticion

    .then (resp => resp.json())
    .then( ({data})=>{
        const {url} = data.images.original.url;

        const img = document.createElement('img')
        img.src= url;
        document.body.append( img )



    } )
    .catch (console.warn)
```

ASYNC AWAIT

La estructura de una promesa es esta:

```
const getImagenPromesa = () =>{
    const promesa = neew Promise(( resolve, reject )=>{
        resolve('http:lsjdhsoaudh.com')
    })
    return promesa
}
```

- Este código se puede simplificar mucho

```
const getImagenPromesa = ()=>
    new Promise( resolve => resolve('http:lsjdhsoaudh.com'))
```

```
getImagenPromesa().then( console.log)
```

- Pero aún asi no es fácil. Con el async y await la cosa cambia
- Esta función normal:

```
const getImagen = ()=>{
    return 'http:lsjdhsoaudh.com'
}
```

- Usando el async obtengo el mismo resultado que la promesa anterior

```
const getImagen =async ()=>{
    return 'http:lsjdhsoaudh.com'
}

getImagen().then(console.log)
```

- Await dice: espera a ejecutar esta línea, antes de ejecutar la siguiente.
    - El async puede ser independiente pero await depende del async
    - Desestructuro la data para no poner data.data

```
const getImagen = async ()=>{
    const apiKey = '1234abc';
    const resp = await fetch(`http:endpoint.random?apikey=${ apiKey}`;
    const {data} = await resp.json();
    )

    //el mismo código para crear la imagen en el html

    const img = document.createElement('img')
        img.src= url;
        document.body.append( img )

}
getImagen()
```

- Para manejar el error hay que utilizar el try y el catch
- Se verá posteriormente

```
    const getImagen = async()=>{

        try{

            const apiKey = '1234abc';
            const resp = await fetch(`http:endpoint.random?apikey=${ apiKey}`);
            const {data} = await resp.json();


            //el mismo código para crear la imagen en el html

            const img = document.createElement('img')
                img.src= url;
                document.body.append( img )

    }catch(error){
        console.error(error)
    }
}
getImagen()
```

Importación y exportación

- Carpeta data/heroes.js con un arreglo de objetos con id, nombre y owner
- Si quiero exportarlo debo usar la palabra export

```
export heroes = [
    {id: 1,
    nombre: 'Spiderman',
    owner: 'Marvel'},

    {id:2,
    nombre: 'Superman',
    owner: 'DC'

    }
    {id:3,
    nombre: 'Flash',
    owner: 'DC'

    }
]
```

- Para importar este archivo, no hace falta el 'js' al importarlo en el archivo destino

```
import {heroes} from './data/heroes'
```

- El snippet es imp
- Puedo eliminar el return y las llaves en la función de flecha

```
    const getHeroeById(id)=> heroes.find(heroe=> heroe.id === id)

getHeroeById(2)
```

- Para elegir varios uso el .filter

```
getHeoresByOwner = (owner) => heroes.filter(owner, (heroe)=>heroe.owner === owner)

getHeroesByOwner('DC')
```

Multiples exportaciones e importaciones

- Podría exportar heroes por defecto , ir al final del archivo y export default NombreDelArchivo
- Si quiero exportar alguna constante solo debo poner la palabra export
- Cuando uso una exportación por defecto y otra individual, uso la desestructuración

```
    {id:3,
    nombre: 'Flash',
    owner: 'DC'

    }

    export const owners= ['DC', 'Marvel']



//y en el archivo destino


    export default heroes, {owners} from './data/heroes'
```

- Pueden hacerse todas las exportaciones por defecto a la vez y usar la desestructuración

```
//Al final del archivo origen

export{
    heroes, //si pusiera heroes as default, la importación en el archivo destin
sería como la anterior
    owners
}

//Al principio del archivo destino

import{heroes, owners} from './data/heroes'
```

Componentes

- Así luce un componente trabajando con la librería react.min.js, react.dom.min.js, babel.min.js

```
<script type="text/babel">

    const divRoot = document.querySelector('#root')
    const nombre = "Goku"

    const h1Tag = <h1>Hola, soy {nombre} </h1>;

    ReactDOM.render(h1Tag, divRoot)
</script>
```

- Se puede ver como selecciono la etiqueta #root, que es un id
- Encapsulo la etiqueta h1 en h1Tag y con las llaves le añado texto del lado de javascript dentro de la etiqueta h1
- Para finalizar renderizo con ReactDOM, seleccionando "el qué y dónde"; la constante donde guardé el Tag, el qué, y el id, dónde
- Todo es un componente: el menú, formulario, cada item, son segmentos
    - **piezas de código encapsuladas que tienen una acción específica**
    - Pueden contener unos de otros, *padres e hijos*.
    - Pueden **tener un ESTADO o NO**

¿Qué es un estado?

- Cuando un formulario es renderizado por primera vez, tiene un *ESTADO INICIAL*.
    - Es como se encuentra la información por primera vez
- Los campos no tienen ningún valor
- En el momento que los campos, desde la primera tecla que introdujo un caracter, son rellenados, **CAMBIA EL ESTADO**

```
EL ESTADO es como se encuentra la información de un componente en un punto
determinado del tiempo
```

Primera aplicación

Escribir este comando para iniciar

```
$}npx create-react-app nombre-de-la-aplicación
$}cd nombre-de-la-aplicacion
$}npm start
```

Hola Mundo en React

- Borro toda lo que lleva la carpeta src

```
El lenguaje de HTML que se usa del lado de javascript se llama JSX
Para usarlo, hay que realizar la importación de React
Para renderizar hay que usar el ReactDOM, por lo que hay que importarlo
```

```
import React from 'react'
import ReactDOM from 'react-dom'

const saludo = <h1>Hola Mundo</h1>
```

- En el index.html, en el body, tengo un div con el id "root"

```
<body>

<div id="root">

</div>

</body>
```

- Necesito crear la referencia a esa etiqueta para renderizar saludo
- Lo haré por el id con querySelector
- Utilizo ReactDOM y entre paréntesis, primero el qué y luego el dónde

```
import React from 'react'
import ReactDOM from 'react-dom'

const saludo = <h1>Hola Mundo</h1>

const divRoot = document.querySelector('#root');

ReactDOM.render(saludo, divRoot)
```

- Creo un archivo llamado PrimerApp.js
- Hay dos tipos de componentes, basados en clases y basados en funciones
- En este curso se trabaja con los *Functional Components*
- Al usar JSX hay que importar react, con el snippet "imr"

```
import React from 'react
const PrimeraApp = () => {
    return <h1> Hola Mundo!</h1>
}

export default PrimeraApp;
```

- Yo puedo importarla al index.js y usarla

```
import React from 'react'
import ReactDOM from 'react-dom'
import PrimeraApp from '../PrimeraApp'

const divRoot = document.querySelector('#root');

ReactDOM.render(<PrimeraApp  />, divRoot)
```

El reotrno de los elementos se hacen en un div contenedor o Fragment

Si escribo la etiqueta vacía tampoco hay div adicional y funciona igual que un Fragment

- Importo el elemento Fragment de forma independiente

```
import React, { Fragment } from 'react'
//ejemplo1

const PrimeraApp = ()=>{

    return(

        <Fragment>
            <h1>Hola Juanola</h1>
            <h2>Adiós Juan</h2>
        </Fragment>
    )
}

//ejemplo2: etiqueta vacía, no hay div
<>
    <h1>Hola Juanola</h1>
```

```
    <h2>Adiós Juan</h2>
</>

//ejemplo3: div
    <div>
    <h1>Hola Juanola</h1>
    <h2>Adiós Juan</h2>
    </div>

# Imprimir variables en el html

- Las llaves no aceptan todo tipo de elementos
~~~~javascript

const PrimeraApp = ()=>{
    const saludo = "Hola, Juan"
    return(

        <Fragment>
            <h1>{saludo}</h1>
            <h2>Adiós Juan</h2>
        </Fragment>
    )
}
```

```
   NO IMPRIME CONDICIONALES, BOOLEANS NI OBJETOS


- Pero puedes convertir el objeto con JSON.Stringify y la etiqueta pre
- Mas info modzilla
```

```
<pre>{JSON.Stringify(saludo, null, 3)}</pre>
```

Comunicación entre componentes---> Props

- En este ejemplo, el componente padre sería PrimeraApp.
- Puedo pasar info a través de las props a otros componentes
- Se pueden ver las props ,no tiene de momento, en el navegador

```
const PrimeraApp=(props)=>{
    const saludo = "Hola Mundo!"

    console.log(props)
```

```
    return (
        <>
         <h1> {saludo}</h1>
         <h2>Mañana te veo</h2>
        </>
    )

}
```

- Este console.log(props) devuelve, imprime en consola, un objeto vacío no un undefined!

```
import React from 'react'
import ReactDOM from 'react-dom'
import PrimeraApp from '../PrimeraApp'

const divRoot = document.querySelector('#root');

ReactDOM.render(<PrimeraApp saludo= "Hola, soy Leonard Cohen" />, divRoot)
```

- Ahora el console.log devuelve al Leonard C.
- Se pueden ver en el navegador e incluso agregarlas *in situ*, el console.log las imprimirá
    - Esto es muy util para probar cosas desde el navegador sin refrescar.

¿Cómo lo haría si no tuviera ese const saludo?

- Haría algo así:

```
const PrimeraApp=(props)=>{


    console.log(props)

    return (
        <>
            <h1> {props.saludo}</h1>
            <h2>Mañana te veo</h2>
        </>
    )

}
```

- Pero usando la desestructuración quedaría así:

```
const PrimeraApp=({saludo})=>{
```

```
    return (
        <>
          <h1> {saludo} </h1>
          <h2>Mañana te veo</h2>
        </>
    )

}
export default PrimeraApp
```

Podría pasar que no se mandara esa propiedad saludo. Se podría definir un valor por defecto, o indicar que tiene que haber un valor en props

PropTypes

```
import React from 'react'
import  PropTypes from 'prop-types'


const PrimeraApp=({saludo})=>{

    return (
        <>
          <h1> {saludo} </h1>
          <h2>Mañana te veo</h2>
        </>
    )

}

PrimeraApp.propTypes = {
    saludo: PropTypes.string //esto obliga a que sea de tipo String
}


export default PrimeraApp
```

- Si yo pusiera saludo = 123, saltaría error
- Para hacer que sea una propiedad obligada, pues sin ello no daría error si no hubeira argumento:

```
PrimeraApp.propTypes = {
    saludo : PropTypes.string.isRequired,
    otra: PropTypes.number  //si quisiera poner otra, se separan por comas
}
```

DefaultProps

- Añado otra prop subtitulo y lo pinto en el h2, si es por defecto lo usual es definirlo directamente
- La prop no aparece en la consola del navegador

```
import React from 'react'
import  PropTypes from 'prop-types'


const PrimeraApp=({saludo, subtitulo="Soy un subtitulo"})=>{

    return (
        <>
          <h1> {saludo} </h1>
          <h2> {subtitulo} </h2>
        </>
    )

}

PrimeraApp.propTypes = {
    saludo: PropTypes.string //esto obliga a que sea de tipo String

}
```

- hay otra manera de hacerlo, en las propTypes

```


PrimeraApp.propTypes = {
    saludo: PropTypes.string //esto obliga a que sea de tipo String

}

PrimeraApp.defaultProps = {
    subtitulo: 'Soy un subtítulo'
}
```

Se puede dejar el objeto vacío y que aparezca en las props para documentar que es necesario, obligatorio

Pruebas a Componentes

- Pruebas a primeraApp. Las importaciones necesarias. El saludo era requerido.

```
const divRoot = document.querySelector('#app')

ReactDOM.render(<PrimeraApp saludo= 'Hola, soy Christian'/>, divRoot)
```

- Varias cosas se pueden comprobar:
    - que renderice bien el h1, el subtitulo, etc

```
const primeraAppp = ({saludo, subtitulo}) => {

    return(
        <>
            <h1>{ saludo }</h1>
            <p>{ subtitulo }</p>

        </>
    )
}

PrimeraApp.propTypes = {
    saludo: PropTypes.string.isRequired
}

PrimeraApp.defaultProps = {
    subtitulo: "Soy un subtitulo"
}
```

- Creo el archivo PrimaraAPp.test.js, uso el describe
- Necesito renderizar el componente y evaluar sobre el producto renderizado
- Creo un contenedor wrapper e importo render y react, ya que renderizo con JSX
- Saludo es requerido, sin saludo dará error

```
import {render} from '@testing-library/react'
import PrimeraApp from '.....'
import React from 'react'


test('debe de mostrar Hola soy Christian', ()=>{
    const saludo = 'Hola, soy Christian"
    const wrapper = render(<PrimeraApp saludo={saludo}/>)
```

```
    wrapper
})
```

- Wrapper tiene un montón de métodos, aquí se usará con enzyme, el método getByText.
- De aquí en adelante se usará enzyme, no la librería @testing de React
- Se puede usar en la desestructuración

```
test('debe de mostrar Hola soy Christian', ()=>{
    const saludo = 'Hola, soy Christian"
    const {getByText} = render(<PrimeraApp saludo={saludo}/>)

    expect.getByText(saludo).toBeInTheDocument();
})
```

- Esto da error porque no esta configurado en setupTest para extender el expect

Enzyme Testing Unit

- ENZIME
- Paso 1. Inst JEST en el proyecto: npm install --save-dev jest
- P2. Inst la versión de enzime para REACT 17: npm install --legacy-peer-deps --save-dev @wojtekmaj/enzyme-adapter-react-17
- P3. Add a setupTests.js:

  import Enzyme from 'enzyme'; import Adapter from '@wojtekmaj/enzyme-adapter-react-17';

  Enzyme.configure({ adapter: new Adapter() });

- P4. Inst las dependencias faltantes:

npm install --save-dev enzyme

- P5. Inst enzyme-to-json:

npm install --save-dev enzyme-to-json

- P6. Add a setupTest.js:

  import {createSerializer} from 'enzyme-to-json'; expect.addSnapshotSerializer(createSerializer({mode: 'deep'}));

```
npm install --save-dev enzyme @wojtekmaj/enzyme-adapter-react-17
```

- Creo el archivo en src

setuptests.js:

```
import Enzyme from 'enzyme';
import Adapter from '@wojtekmaj/enzyme-adapter-react-17';
import {createSerializer} from 'enzyme-to-json';

Enzyme.configure({adapter: new Adapter() });
expect.addSnapshotSerializer(createSerializer({mode:'deep'}))
```

```
import {shallow} from 'enzyme'
import'@testing-library/jest-dom'

test('debe de mostrar PrimerApp correctamente', ()=>{
     const saludo = "Hola, soy Christian"

    const wrapper = shallow(<PrimeraApp saludo={saludo}>)

    expect(wrapper).toMatchSnapshot();

})
```

- El Snapshot no se toca manualmente, pero se puede abrir para ver que hay
- Es una fotografía de lo renderizado.
- Si dice ShallowWrapper no es lo que ineteresa.
    - Por eso la extensión configurada anteriormente

Revisar elementos dentro del componente

- Para comprobar si a mi componente le mandamos un subtitulo
- Tocar u para actualizar el snapshot
- Wrapper contiene toda el html relacionada con lo renderizado en el snapshot: los h1, los p, etc
- Lo puedo barrer con el .find, como si usara un querySelector
- .text me devuelve el texto del p. Ahora hay que comprobar con el expect que el texto sea igual que el subtitulo

```
test('debe de mostrar  el subtitulo enviado por props', ()=>{
    const saludo = 'Hola, soy Christian';
    const subtitulo = "Este es un subtitulo"


    const wrapper = shallow(<PrimeraApp
                            saludo={saludo}
                        subtitulo = {subtitulo}/>)
    const textoParrafo = wrapper.find('p').text()
```

```
expect(textoParrafo). toBe( subtitulo );

})
```

Pruebas CounterApp

```
import React, {useState} from 'react'
import PropTypes from 'prop-types'




const CounterApp = ({value = "10"}) =>{

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



       const handleAdd =() =>{
            setCounter(c=> counter +1)

       }
       const handleSubstract = ()=>{
           setCounter((c)=> c -1)
       }
       const handleReset = ()=>{
           setCounter(c => c =value)
       }

    return(
        <>
            <h1>CounterApp</h1>
            <h2>{counter}</h2>
            <button onClick={handleAdd}>+1</button>
            <button onClick={handleSubstract}>-1</button>
            <button onClick={handleReset}>RESET</button>

        </>
    )


}


CounterApp.propTypes = {
    value: PropTypes.number
```

```
}

export default CounterApp;
```

- Debe mostrar CounetrApp correctamente
- Debe de mostrar el valor por defecto de 100. Usar el wrapper.find tomando del html donde muestra el contenido
- Importar React, shallow y CounterApp

```
test('debe de mostrar CounterApp', ()=>{
    const wrapper = shallow(<CounterApp  value = 10 />)
    expect(wrapper).toMatchSnapshot(;)

})

test('debe de mostrar el valor por defecto de 100', ()=>{

    const wrapper = shallow(<CounterApp  value = 10 />)

    const counterText = wrapper.find('h2').text().trim
    //console.log(counterText)

    expect(counterText).toBe('100')

})
```

- El resultado donmde renderiza el numero esta en un h2 y solo hay uno
- Para React los espacios dentro de las llaves son espacios también

---

Simular Eventos- Click

- Puedo crear el wrapper dentro del describe, un nivel superior , para poder usarlo en los otros tests
- at 0 es elegirlo en base a su poción índice
- Para encontrar el valor es la misma sentencia
- esperaía un 11 ya que el valorp or defecto es 10 y le he añadido 1

```
    const wrapper = shallow(<CounterApp  value = 10 />)

test('Debe de incrementar contador con el botón +1', ()=>{

    wrapper.find(button).at(0).simulate('click')
    const counterText = wrapper.find('h2').text().trim
    expect( counterText).toBe('11')
```

```
 })
test('Debe de restar contador con el botón -1', ()=>{

    wrapper.find(button).at(1).simulate('click')
    const counterText = wrapper.find('h2').text().trim
    expect( counterText).toBe('9')

 })
```

- No recibe 9 del toBe, recibe 10 Porque las pruebas son ejecutadas UNA DETRÁS DE LA OTRA
- Se pueden reinicializar antes de que se ejecuten cada una de ellas, ya que tienen un ciclo de vida
- Para ello usaré el beforeEach()
- Cambio el wrapper a let para que no sea una variable de scope

```
describe('Pruebas en el CounterApp', ()=>{

    let wrapper = shallow(<CounterApp  value = 10 />)

    beforeEach(()=>{
        wrapper = shallow(<CounterApp  value = 10 />)

    })

})
```

- el reset recibe el valor que se le manda por las props

```
test('debe de colocar el valor por defecto con el btn reset')

    const  wrapper = shallow(<CounterApp  value = 105 />)

    wrapper.find().at(0).simulate('click')

    expect( counterText).toBe('106')

    wrapper.find().at(2).simulate('click')
    expect( counterText).toBe('105')
```

Snippets

- rafce ---> functional component con importacion de React
- rafcp ---> functional component con importacion de PropType
- imr -----> import React
- imd -----> import ReactDOM

CounterApp

Eventos

- Tengo tres componentes que lucen así: index.html, index.js y CounterApp.js

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CounterApp</title>
</head>
<body>

  <div id="app">

    <CounterApp value ={10} />


    </div>

  <script>

  </script>

</body>
</html>
```

```
import React from 'react'
import ReactDOM from 'react-dom'
import CounterApp from './CounterApp'

const divRoot = document.querySelector('#app');

ReactDOM.render(<CounterApp value={ 10 } />, divRoot)
```

```
import React from 'react'
import PropTypes from 'prop-types'


const CounterApp = ({value}) =>{
    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button>+1</button>

        </>

    )


}


CounterApp.propTypes = {
    value: PropTypes.number
}

export default CounterApp;
```

En la documentación oficial se pueden consultar todos los eventos

```
        es.reactjs.org
```

- Cómo lo que busco es la acción del click en el botón usaré onClick
- Para introducir código de jaascript usaré las llaves { }
- Usaré una función de flecha e imprimiré el evento para inspeccionar

```
import React from 'react'
import PropTypes from 'prop-types'


const CounterApp = ({value}) =>{
    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={(e)=>{
                console.log(e)
            }}>+1</button>
```

```
        </>
    )
}


CounterApp.propTypes = {
    value: PropTypes.number
}

export default CounterApp;
```

- Se puede sacar del return esta función
- Cuando el primer argumento de una función es enviado como primer argumento de la función que está dentro: -se puede dejar solo la referencia a la función que quiero ejecutar - el event de onClick va a ser pasado como único argumento del handleAdd

```
const CounterApp = ({value}) =>{

    handleAdd =(e) =>{
        console.log(e)
    }


    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={handleAdd}>+1</button>

        </>
    )
}
```

Porqué no invoco la funcion handleAdd?

- Toda función que no tiene un return especifico retorna undefined
- Todas las funciones retornan algo,
    - Si la invoco en el html entre llaves con un 1 en el return, marcará error:
        - Esperaba una función y recibió un valor numérico
    - Si pongo en el return el mismo console.log y la invoco entre llaves en el html SI funcionaría
    - Serviría para procesar cualquier instrucción y mandar lo que se quiera del click

      Esta es la diferencia de colocarlo con paréntesis o no

useState- Hook

Referencia

- El argumento es el mismo que el primer valor del return, el segundo es una función.
    - Esta función se suele llamar setAlgo. Es utilizada para establecer el valor al primer argumento del return
- Se hace la desestructuración o se pone state directamente

```
const useState = (valor) =>{
    return [valor, ()=>(console.log('Hola Mundo'))]
}

const [nombre, setNombre ] = useState('Bonifacio')
        //state//

console.log( nombre );
setNombre();
```

- Los hooks suelen empezar por use

```
import React, {useState} from 'react'
importPropTypes from 'prop-types'


const CounterApp = ({value}) =>{

    const state = useState('Bonifacio')
    console.log(state) ////---> devuelve un arreglo de 2 items uno el nombre y
otro un dispatchAction


    handleAdd =(e) =>{
        console.log(e)
    }


    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={handleAdd}>+1</button>

        </>
    )
}
```

Se suele usar la desestructuración

- El console.log del state y setState devuelve un arreglo con 2 valores
    - el estate
    - un dispatchAction

```
import React, {useState} from 'react'
importPropTypes from 'prop-types'


const CounterApp = ({value}) =>{

    const [counter, setCounter] = useState(0)


    handleAdd =(e) =>{
        setCounter( counter +1 )
    }


    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={handleAdd}>+1</button>

        </>
    )
}
```

- Si pongo counter ++ me dará error. Porqué?
    - Estoy intentando cambiar el valor a una constante y eso no está permitido. Sería como poner counter = counter + 1
    - Con let tampoco funcionará

      REACT obliga a no mutar el state de esta manera, tiene funciones específicas para modificar el state
- El setCounter le dice a react que el counter cambió, por lo que hay que renderizar de nuevo el componente
    - Pero solo redibujará lo que cambia
- Pudiera ser que no tuviera acceso al counter anterior.
    - En el useState puedo recibir el valor del anterior counter de la desestructuración, y llamarlo con una función de flecha

```
const CounterApp = ({value}) =>{

    const [counter, setCounter] = useState(0)


    handleAdd =(e) =>{
        setCounter( (c) => c++)
    }


    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={handleAdd}>+1</button>

        </>
    )
}
```

HandleSubstract y HandleReset

- Creo 2 botones adicionales en el html
    - Substract y Reset
- Puedo crear la lógica en los botones directamente

```
const CounterApp = ({value}) =>{

    const [counter, setCounter] = useState(0)


    handleAdd =(e) =>{
        setCounter( (c) => c++)
    }
    handleSubstract = ()=>{
         setCounter((c)=>c -1)
    }


    return(
        <>
            <h1>CounterApp</h1>
            <h2>{value}</h2>
            <button onClick={handleAdd}>+1</button>

        </>
    )
```

```
}
```

- Para hacerlo en los botones sería de esta forma:

```
<button onClick={ ()=> setCounter( value)}> RESET</button>
<button onClick={ ()=> setCounter( counter -1)}> Substract</button>
```

Gif Expert 1

- Para consumir la API necesito una APIKey y la documentacion para el endpoint y la data
- Las herramientas de React del navegador también ayudará
- nopx create-react-app gif-expert
- Borro toda la carpeta src
- Incluyo, añado extend-expect a la linea de @testing en setupTsts.js

```
import '@testing-library/jest-dom/extend-expect'
```

El archivo index.js ahora luce así:

```
import React from 'react'
import ReactDOM from 'react-dom'

ReactDOM.render(
    <App />,
    document.getElementById('root')
)
```

- App no está definida y da error pero ahora es correcto
- Creo el componente **GifExpertApp** con un h2 y abajo un hr, una linea de separación
- Cambio App por GifExpertApp en el index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import {GifExpertApp} from '.../...'

ReactDOM.render(
    <GifExpertApp />,
    document.getElementById('root')
)
```

- Añado unos estilos de CSS (avoid)
- La estructura de archivos es importante. Hay documentación sobre ello
- Crearé una lista de categorías para pintarla en pantalla
- para ello necesitaré ol y li y algo que retorne uno (o más valores)
- El ciclo for no va a funcionar, necesito un método, en este caso .map
- Error: Each child needs an unique key prop

- Asignar el valor indice como key puede ser inestable a los cambios, pero viene implicito en el .maps y de momento no hay id
- Esa key es lo que React necesita para saber que objeto está iterando
- Usaré category como key y como valor a mostrar
- Creo un botón de Agregar

```
import React from 'react'

const GifExpertApp = () => {
    const categories = ['One Punch', 'Samurai X', 'Megaman']

  return (
    <>
        <h1>GifExpertApp</h1>
        <hr />
        <button>Agregar</button>


        <ol>
          {categories.map(category =>{
                return <li key= {category}>{category}</li>
          })}
        </ol>
    </>
  )
}

export default GifExpertApp
```

```
CREAR UN ARREGLO CON CONST PARA ALGO DINÁMICO NO VA A FUNCIONAR!!!!
```

USO useState

(LO IMPORTO) - setCategories es la función que voy a llamar para añadir categorias al arreglo - Con el .push NO ES LA MANERA, ya que lo agregaría al arreglo pero no cambió en el State pq NO SE CAMBIA ASI EL STATE - Ademas, es una constante. - Cuando el setCategories maneje el cambio, hará que React vuelva a renderizar el componente y guardará los cambios - Category es el valor, y el setCategories es lo que voy a llamar para añadir categorías - Cuando se usa el setCategories, estoy "cayendo encima" al estado anterior, con lo cual sin el spread sobreescribiría el anterior state dando ERROR - Por ello uso el spread, para extraer las categorías, y añado el nuevo valor -Si lo pongo antes del spread, lo añadirá en primera posición.

```
-Añado el handleAdd al onClick del button
- RESUMEN: Barrido con el .map y pintar un li con category como key
```

```
  - Con el setCategories le caigo encima al state, con el .push (MALA PRAXIS)
reescribo
  - Por ello uso el useState con desestructuración  para añadir item a categories
```

```
import React, {useState} from 'react'

const GifExpertApp = () => {
    const [categories, setCategories] = useState(['One Punch', 'Samurai X',
'Megaman'])

    const handleAdd = ()=>{
        SetCategories(...categories, 'HunterXHunter' );
    }


  return (
    <>
        <h1>GifExpertApp</h1>
        <hr />
        <button onClick={handleAdd}>Agregar</button>


        <ol>
          {categories.map(category =>{
                return <li key= {category}>{category}</li>
          })}
        </ol>
    </>
  )
}

export default GifExpertApp
```

- Otra manera de hacerlo sería con un callback
- El setCategories puede tener un callback, en el cual el primer argumento es EL ESTADO DEL STATE ANTERIOR - + EL NUEVO ESTADO QUE ESTOY AGREGANDO

```
const handleAdd = () =>{

    setCategories(cats =>{...cats, 'HunterXHunter'})

}
```

- Al usar categories como id, si aprieto varias veces el botón se repite el key lo que da ERROR
- Normalmente son id's de bases de datos, no puede ser el índice
- El handleAdd era solo para fines didácticos. BORRADO

Componente AddCategory

- Creo un nuevo directorio llamado components y el archivo de AddCategory
- Quiero un nuevo componente que tenga una caja de texto y al presionar enter agregue una nueva categoría
- Coloco el AddCategory como componente debajo del h2 del GifExpertApp
- Necesito algo que me sirva para añadir categorías: un input
- Uso el useState, lo importo
- setInputValue es lo que voy a usar para cambiar la caja de texto

```
import React, { useState } from 'react'
import ReactDOM from 'react-dom'


const AddCategory = () => {

  const [inputValue, setInputValue]= useState('Hola Mundo')

  return (
    <>
    <input
      type="text"
      value = {inputValue}>


    </>
  )
}

export default AddCategory
```

- En este momento no me deja escribir en la caja, porque este valor no puede cambiar (estado) porque no maneja el onChange.
- Puesto el onChange, necesito extraer ese valor de la caja de alguna manera. Pruebo un console.log del evento
- con el console.log del evento del onChange con un callback puedo ver en consola que debo atacar al e.target.value y pasarlo al handleInputChange

```
onChange = {(e)=>console.log(e)}
```

- Veo que el valor está en e.target.value, mando a llamar el setInputValue con el valor de e.target.value , primero puedo hacer un console.log
- Puedo pintar el inputValue, siempre será el último valor que la persona escribe a tiempo real en pantalla
- Puedo meterlo todo en un form, entonces no hace falta el fragments.

- Manejo el onChange con el handleInputChange, se actualizará cada vez que el texto de la caja cambie
- Quiero manejar el enter para disparar la acción.

```
import React, { useState } from 'react'
import ReactDOM from 'react-dom'


const AddCategory = () => {

  const [inputValue, setInputValue]= useState('Hola Mundo')

  handleInputChange = (e) =>{
    //console.log(e.target.value)
    setInputValue(e.target.value)
  }


  return (
    <>
    <h1>{inputValue}</h1>
    <input
      type="text"
      value = {inputValue}
      onChange={(handleInputChange)} />

    </>
  )
}
```

- Quiero manejar el Submit
- Me interesa prevenir el comportamiento de refresh que tiene el navegador cuando doy Enter al formulario
- Borro el h1, no interesa

```
import React, { useState } from 'react'
import ReactDOM from 'react-dom'


const AddCategory = () => {

  const [inputValue, setInputValue]= useState('Hola Mundo')

  handleInputChange = (e) =>{
    //console.log(e.target.value)
    setInputValue(e.target.value)
  }
  const handleSubmit = (e)=>{
    e.preventDefault()
  }
```

```
  return (

    <form onSubmit = {handleSubmit}>
    <input
      type="text"
      value = {inputValue}
      onChange={(handleInputChange)=>} />

    </form>
  )
}
```

Comunicación entre componentes

- Tenemos un componente encargado de manejar la información del input, AddCategory, y otro de renderizarla, GifExpertApp
- Para añadir un elemento, el setCategories lo tengo en el GifExpertApp
- Para llamar al setCategories en el AddCategories, SE LO PUEDO PASAR COMO PROPS mandándole de referencia del setCategories
- Ahora la puedo ver en Componentes del Browser, en las props setCategory, la función
- Debo llamarla en el handleSubmit!! Cómo lo hago? Hay que recibirlo de las props

```
import React, {useState} from 'react'
import AddCategories from '....'


const GifExpertApp = () => {
    const [categories, setCategories] = useState(['One Punch', 'Samurai X',
'Megaman']);



  return (
    <>
        <h1>GifExpertApp</h1>
        <AddCategories  setCategories= {setCategories}/>
        <hr />
        <button onClick={handleAdd}>Agregar</button>


        <ol>
          {categories.map(category =>{
                return <li key= {category}>{category}</li>
          })}
        </ol>
```

```
    </>
  )
}
```

Uso la desestructuración en lo que sería el props como argumento del AddCategory

```
//Esto!!! desestructuración
const AddCategory =({setCategories})=>{}

//en lugar de . No se lleva poner props. luego habría que escribir props. todo
const AddCategory = (props) =>{}
```

- Lo que sea que yo envíe como props al setCategories de AddCategory, es lo que yo obtengo como props en el componente, en este caso el setCategories
- He de llamar al setCategories con un callback para recibir el estado anterior y modificarlo con el nuevo state
- Se hizo anteriormente como ejemplo, desestructurando categories y añadiendo el nuevo state como segundo valor
- Bien podría llamar las categoría desde la desestructuración de las props, pero puedo hacerlo con un callback
- El onsubmit es en el form, no en el input.
- tengo muteado todo el rato el handleAdd y el setCategories.
- He de hacer una validación para manejar errores en el handleSubmit, que conecto con el onSubmit, ahora si borro todo inserta un espacio en blanco y eso no debería ser
- Si tiene mas de dos caracteres llamo al setCategories y reseteo el inputValue mandando un string vacío

```
import React, { useState } from 'react'



const AddCategory = ({setCategories}) => {

  const [inputValue, setInputValue]= useState('')

  handleInputChange = (e) =>{
    //console.log(e.target.value)
    setInputValue(e.target.value)
  }
  const handleSubmit = (e)=>{
    e.preventDefault();

   if(inputValue.trim().length >2{
      setCategories(cats => [...cats, inputValue])
      setInputValue('')
```

```
    })
  }
  setCategories(cats => [...cats, inputValue])


  return (

    <form onSubmit = {handleSubmit}>
    <input
      type="text"
      value = {inputValue}
      onChange={(handleInputChange)=>} />

    </form>
  )
}
```

- Uso el propTypes para obligar a usar el setCategories. Lo importo

```
import React,  {useState } from 'react'
import PropTypes from 'prop-types'


const AddCategory = ({setCategories}) => {

  const [inputValue, setInputValue]= useState('')  //no pone hola mundo!!! Sin el
string vació ' ' da undefined= error

  handleInputChange = (e) =>{
    //console.log(e.target.value)
    setInputValue(e.target.value)
  }
  const handleSubmit = (e)=>{
    e.preventDefault();

   if(inputValue.trim().length >2{
      setCategories(cats => [...cats, inputValue])
      setInputValue('')

    })
  }
  setCategories(cats => [inputValue, ...cats]) //coloco el inputValue primero para
que lo
                                                // muestre primero


  return (

    <form onSubmit = {handleSubmit}>
    <input
      type="text"
      value = {inputValue}
```

```
        onChange={(handleInputChange)=>} />

      </form>
    )
}

AddCategories.propTypes = {
  setCategories: PropTypes.func.isRequired
}
```

- Ahora da un error, pero es apropósito, hecho con fines didácticos
- En el useState borré el hola mundo, eso da error. Si el estado actual da undefined, inputValue es undefined. ERROR
- Se requiere que el valor inicial del useState sea algo,un string, iniciarlo con un string vacío

```
const [inputValue, setInputValue] = useState('')
```

Fetch API-> obtener las imagenes deseadas

- Mi objetivo ahora es crear un nuevo componente, que cuando note que hay un elemento
    - Haga la petición http
    - Traiga las imágenes correspondientes a la categoría
    - Y se desplieguen en pantalla
- Lo primero es crear un nuevo componente que reciba la info de la caja para procesarla petición http
- Para realizar el ejercicio se pondrá un valor al inicio del useState 'OnePunch', donde se inició el string vacío
- Debo pasar en las props category haciendo desestructuración en GifGrid y pasarlo como props en GifExpertApp PARA HACER LA CONEXIÓN

```
const [categories, setCategories] = useState('OnePunch')
```

- Lo llamo GifGrid

```
import React from 'react'
import ReactDOM from 'react-dom'

export const GifGrid = ({category}) => {
  return (
    <div>
```

```
      <h3>{category}</h3>
    </div>
  )
}

~~
- Quito el botón, reacomodo el código
- Pinto el GifGrid con la categoría que estoy evaluando en este mismo momento,
coloco el key obligatorio

~~~~js
import React, {useState} from 'react'
import {AddCategories} from '....'
import {GifGrid} from '......'

const GifExpertApp = () => {
  const[categories, setCategories] = useState(['One Punch']);



  return (
    <>
        <h1>GifExpertApp</h1>
        <AddCategories  setCategories= {setCategories}/>
        <hr />



        <ol>
          {categories.map(category =>

            <GifGrid
              key: {category}
              category = {category} />

          )}
        </ol>
    </>
  )
}
```

- Si miro en componentes del navegador debería aparecer GifGrid: OnePunch
- Si agrego otro en el input y doy enter, aparece una nueva key del GifGrid con la categoría que haya escrito
- ENTONCES NECESITO RECIBIR UNA CATEGORÍA Y HACER LA PETICIÓN HTTP
- Necesito el endpoint de la API, getURL, consultar en la web
- Uso Postman
- Introduzco la URL del endpoint del SEARCH seguido de api_key, mi apiKey extraida de la web

- Recibo una respuesta con toda la data
- No devuelve nada porque no hay consulta
- Para ello, en params query escribo q en la izquierda y busco lo que haya en consola, DragonBall, lo pongo en la casilla de Value a la derecha
- Ahora si aparece toda la data de 20 imágenes de DragonBall
- Puedo poner en key de postman limit y de value 10, para manejar mejor la info
- api_key debe de ser el último valor, abajo de todo de key
- en consola debería aparecer type, y todo tipo de valores de cada gif

---

- Ahora creo una constante llamada getGifs con esa url -copy+past y le añado https:
- Ahora hay que llamar al endpoint
- Hago la función getGifs async
- Hago la respuesta del fetch await
- Guardo el json
- A mi lo que me nteresa es la data de la data, uso la desestructuración para trabajarla

```
import React from 'react'
import ReactDOM from 'react-dom'

export const GifGrid = ({category}) => {

  const getGifs= async ()=>{

    const url ='https://api.giphy./1234abc/apiKey567DEF'

    const resp = await fetch( url );
    const {data} = await resp.json(); //desestructuracion!!
    }
  return (
    <div>
      <h3>{category}</h3>
    </div>
  )
}
```

- Ahora puedo barrer todas las imagenes y extraer el id, el título y la url
- Se podría usar la desestructuración para no tener que escribir img.
- Puedo usar el interrogante en images para que si tiene toda esa info la utilice

```
import React from 'react'


export const GifGrid = ({category}) => {
```

```
  const getGifs= async ()=>{

    const url ='https://api.giphy./1234abc/apiKey567DEF'

    const resp = await fetch( url );
    const {data} = await resp.json();

    çconst gifs = data.map(img =>{
        return{
          id: img.id,
          title: img.title,
          url: img.images?.downsized_medium.ulr
        }
    })

    }

getGifs()

  return(
    <div>
    <h3> {category}</h3>
    </div>
  )
}
```

useEffect

```
HAY UNA SERIE DE INCONVENIENTES DE ESTA MANERA DE HACER, POR ESO HABLAMOS DE
useEffect
```

- Importo el useState y lo declaro como contador, count, con el valor de cero
- Yo lo que quiero es llamar al setCount. Todo esto es para entender el useEffect
- Creo un boton

```
import React, {useState} from 'react'


export const GifGrid = ({category}) => {

  const [count, setCount] = useState(0)

  const getGifs= async ()=>{
```

```
    const url ='https://api.giphy./1234abc/apiKey567DEF'

    const resp = await fetch( url );
    const {data} = await resp.json();

    çconst gifs = data.map(img =>{
        return{
          id: img.id,
          title: img.title,
          url: img.images?.downsized_medium.ulr
        }
    })

    }

getGifs();


  return(
    <div>
    <h3> {category}</h3>
    <h3>{count}</h3>
      <button onClick={()=> setCount(coun+1)}>
    </div>
   )
}
```

- Ahora cada vez que aprieto el botón esta disparando una petición http y pintando un numero incremental, porque React notó un cambio
- Entonces vuelve a ejecutar todo el código, porque en medio se encuentra el getGifs,
- Cada vez que detecta un cambio dispara una petición http, porque hay que actualizar las referencias
- Hay un peligro de bucle infinito, podria pasar con el setImagenes.
- Se puede evitar con useEffect. Me va a permitir ejecutar código de manera condicional
- useEffect recibe una función que es lo que quiero ejecutar, y en este caso recibe el getGifs
- El segundo valor que se le manda es una arrgelo de dependencias.
- Si se manda vacío se disparará una única vez
    - Solo se efectuará cuando el componente es renderizado por primera vez

```
import React, {useState. useEffect} from 'react'


export const GifGrid = ({category}) => {

  const [count, setCount] = useState(0)
  useEffect(()=>{
    getGifs()
  }, [])
```

```
  const getGifs= async ()=>{

    const url ='https://api.giphy./1234abc/apiKey567DEF'

    const resp = await fetch( url );
    const {data} = await resp.json();

    çconst gifs = data.map(img =>{
        return{
          id: img.id,
          title: img.title,
          url: img.images?.downsized_medium.ulr
        }
    })

    }

//getGifs();//yo no quiero que este getGifs siga realizando peticiones http cada
vez que renderiza el componente


  return(
    <div>
    <h3> {category}</h3>
    <h3>{count}</h3>
      <button onClick={()=> setCount(coun+1)}>
    </div>
   )
}
```

Mostrar los títulos de las imágenes

- Ahora puedo observar en consola con un console.log que le doy al contador y solo se efectúa una vez la petición http
- Ya no hace falta el contador, LO BORRO
- Dejo el category, añado un ol con 1 item debajo del category de GifGrid
- Uso el useState -En el setImages el nuevo estado serán los gifs con la respuesta fetch
    - Si voy a la pestaña de componentes del navegador veo en consola que tengo el arreglo de imagenes en GifGrid -Reemplazo el item. Hago el barrido con .map y manejo la imagen en un li con la data de la api
- desestructuro img con la data que necesito que es el id y el title, asi no escribo img.id e img.title

```
import React, {useState, useEffect} from 'react'


export const GifGrid = ({category}) => {
```

```
  const [images, setImages] = useState([]);

  useEffect(()=>{
    getGifs()
  }, []) //el arreglo de dependencias vacío, sólo renderiza una vez


  const getGifs= async ()=>{

    const url ='https://api.giphy./1234abc/apiKey567DEF'

    const resp = await fetch( url );
    const {data} = await resp.json();

    const gifs = data.map(img =>{
        return{
          id: img.id,
          title: img.title,
          url: img.images?.downsized_medium.ulr
        }
    })
  console.log(gifs)
  setImages(gifs)

  }


  return(
    <div>
    <h3> { category } </h3>
    <ol>{

      images.map(({id, title})=>(

        <li key= {id}> {title }</li>

      ))
    }</ol>
    </div>
  )
}
```

- Se imprimen en pantalla los 10
- Creo el componente GifGridItem para construir cada elemento a mostrar
- Le pongo img pero es una prop en realidad

```
import React from 'react'


export const GifGridItem = (img) => {
```

```
  cnsole.log(img)

  return (


    <div>GifGridItem</div>
  )
}
```

Voy al GifGrid y quito el ol, para pintar mi GifGridItem, lo importo

```
return(
    <div>
    <h3> { category } </h3>


      images.map(img)=>(
        <GifGridItem
        key= {img.id}
        img = {img}/>

      )

    </div>
  )
```

- Un tip. Uso el spread del img en el GifGrid

```
return(
    <div>
    <h3> { category } </h3>


      images.map(img)=>(
        <GifGridItem
        key= {img.id}
        {...img}/>

      )

    </div>
  )
```

- Imprimo las props en el GifGridItem
- Veo que el resultado es EL MISMO!! puedo ver en consola que recibo el title, la url y el id.

    - Estoy mandando cada una de las propiedades de las imagenes como una propiedad independiente

```
export const GifGridItem = (props) =>{
  console.log(props)

  return(
    <div></div>
  )
}
```

- Uso la desestructuracion y hago algo de carpintería

```
export const GifGridItem = ({ id, title, url}) =>{
  console.log({id, title, url})

  return(
    <div>
      <img src={url} alt={title} />
      <p> {title}</p>
    </div>
  )
}
```

Helpers - getGifs

- El getGifs lo tengo en GifGrid. Es un componente en si mismo, puedo extraerlo para hacer el componente más sencillo y que haga solo una función específica
- Pongo el url en back ticks para hacer un template literal con el category
- Va despues de q="nombre del gif"
- Para evitar errores de espacios se usa encodeURI

```
export const GifGrid = ({category}) => {

  const [images, setImages] = useState([]);

  useEffect(()=>{
    getGifs()
  }, []) //el arreglo de dependencias vacío, sólo renderiza una vez


  const getGifs= async ()=>{
```

```
    const url
=`https://api.giphy./1234abc/apiKey567DEFq=${encodeURI(category)}&limit9uw0nu`

    const resp = await fetch( url );
    const {data} = await resp.json();

    const gifs = data.map(img =>{
        return{
          id: img.id,
          title: img.title,
          url: img.images?.downsized_medium.ulr
        }
    })
  console.log(gifs)
  setImages(gifs)

  }


  return(
    <div>
    <h3> { category } </h3>
    <ol>{

      images.map(({id, title})=>(

        <li key= {id}> {title }</li>

      ))
    }</ol>
    </div>
  )
}
~~
- Creo un nuevo directorio llamado Helpers y un nuevo archivo getGifs
- Le pongo un export antes del const y a correr!
- Le pongo de return gifs, ya que no tengo el setImages
~~~~jsx
export const getGifs= async ()=>{

      const url ='https://api.giphy./1234abc/apiKey567DEF'

      const resp = await fetch( url );
      const {data} = await resp.json();

      const gifs = data.map(img =>{
          return{
            id: img.id,
            title: img.title,
            url: img.images?.downsized_medium.ulr
          }
      })
    return gifs
```

```
    }
```

- El useEffect del GifGrid ya no tiene el getGifs, pero lo puedo importar.
- Este getGifs retorna una promesa, con lo cual puedo usar el .then
- Añado el category como dependencia, porque si esta cambiara , pod´ria suceder que se volviera a disparar la renderización. La consola da un warning al respecto.

```
export const GifGrid = ({category}) => {

  const [images, setImages] = useState([]);

  useEffect(()=>{
    getGifs(category).then(imgs=> setImages(imgs))
  }, [category]) //el arreglo de dependencias vacío, sólo renderiza una vez




  return(
    <div>
    <h3> { category } </h3>
    <ol>{

      images.map(({id, title})=>(

        <li key= {id}> {title }</li>

      ))
    }</ol>
    </div>
  )
}
```

- El useEffect se puede resumir por ser una funcion cuyo primer argumento es mandado como primer argumento quedando asi:

```
useEffect(()=>{
  getGifs( category)
  .then(setImages)
})
```

CUSTOM HOOK

- Un customHook es extraer lógica y hacerla sencilla para ser reutilizada

- el useEffect hace que cuando se cargha el componente por primera vez:
    - lance la petición para obtener los gifs
    - los coloca en las imagenes
- Para los customHooks se recomienda crear un nuevo directorio llamado hooks
- Creo el archivo useFetchGifs.js
- Los hooks empiezan por use. No son más que funciones
- ¿Que diferencia hay con un funcional component? .- Este hook puede tener su estado, y puede indicar a los componentes que lo utyilicen cuando deben renderizarse porque algo cambió.

```
export const useFetchGifs = () =>{
  const [state, setState] = useState({
    data: [],
    loading: true
  })
  return state; //data: [], loading: true, este es el state
}
```

- Almaceno este estado en una constante dentro del GifGrid, quito código y lo importo
- Uso desestructuración

```
export const GifGrid =({category})={

const [data,loading] = useFetchGifs();


return(
  <>
    <h2>{category}</h2>
    {loading ? 'cargando...':'Fin de carga'}

  </>
)
}
```

- Los customHooks pueden tener efectos, pueden usar reducer, contextos
- Cuando se ejecute el GifGrid llama al useFetchGifs
- En UseEffect es donde voy a ejecutar el cuerpo de mi peticion http
- Importo del helpers el getGifs y le paso la categoría como argumento.
- Problema! no la tengo, pero SE LA PUEDO MANDAR POR LAS PROPS DESDE EL GIFGRID

Obtener imágenes y bandera de carga

- Cambios en el GifGrid

```
    export const GifGrid =({category})={

    const [data,loading] = useFetchGifs(category);


    return(
      <>
        <h2>{category}</h2>
        {loading ? 'cargando...':'Fin de carga'}

      </>
    )
    }
~~~
- La recibe en las props;
- Solo efectuará el cambio si la categoría cambia
- GetGifs una promesa, los effects no pueden ser asincronos
-
~~~js
export const useFetchGifs = (category) =>{
  const [state, setState] = useState({
    data: [],
    loading: true
  })

  useEffect(()=>{
  getGifs()
    }[category]) //
}

~~
- Tengo que manejar la info del setState en el mismo orden
~~~js
export const useFetchGifs = (category) =>{
  const [state, setState] = useState({
    data: [],
    loading: true
  })

  useEffect(()=>{
  getGifs(category).
  then( imgs =>{
    setState({
      data: imgs,
      loading: false
    })
```

```
  })
    }[category]) //
}

~~
```

Pruebas Unitarias

- Unitarias: enfocadas en pequeñas funcionalidades
- Integración: cómo funcionan varias piezas en conjunto

- Ejemplo del coche:
  - unitarias: que la llanta gire bien, que sea una llanta, la resistencia es o no es la adecuada
  - integración: cojo 4 llantas, las monto en la carrocería y lo evaluo en conjunto el motor, las ventanas, el chasis, etc

---

- Tienen que ser:
  - Fáciles de escribir
  - Fáciles de leer
  - Confiables
  - Rápidas
  - Principalmente unitarias

Se aplican en tres pasos

AAA

```
- Arrange---> Arreglar: preparamos el estado inical, el sujeto a probar. Se
inician las constantes, las impoprtaciones, etc
- Act-------> Actuar: aplicamos acciones o estímulos al sujeto. Llamar métodos,
simular clics
- Assert----> Afirmar: observar el comportamiento resultante: que algo cambie o no

        Las pruebas no son una prueba de que no haya errores. Estan hechas por
human@s y l@s human@s cometemos errores
        Las pruebas pueden fallar
```

- Las librerías no son lo que se prueba, lo que se preban son las interacciones con esa librería que trabajen como se espera
- No hay que probarlo todo
- Se puede probar lo esencial si hay poco tiempo
- Los conceptos de las pruebas son aplicables en cualquier framework-lenguaje-librería

---

NOTA: en las anotaciones anteriores se usó useState como nombre de la función para explicar su funcionamiento, esto puede dar error. Cambiar a usState

---

- npm i para reconstruir los módulos de node
- npm start para iniciar la app

- Creo la capeta test en src
- Creo un nuevo archivo llamado demo.test.js
- El nombre no es importante, lo importante es la cola .test.js
- Debo hacer las importaciones en el archivo
- Ene l package JSON puedo ver en la seccion scripts, test: react-scripts test, preconfigurado para jest

Jest-Expect- ToBe

- El snippet es test
- El comando para ejecutar es:
  - npm run test
- En el punto 3, la docu oficial anota muchos métodos para el expect Para comparar usaré el toBe

```
test('probando, probando...deben ser iguales los strings', ()=>{
    //punto 1, iniciar

    const mensa = "Hola Mundo!"

    //punto2, aplicar estimulo

    const mensa2: `Hola mundo!`

    //punto3

    expect(mensa).toBe(mensa2);

})
```

- Pasa la prueba
- Al final de lo que escupe la consola, hay varias opciones tecleando w con una tecla, a,f q, p, t, Enter. Usar las flechas para elegir
- Coloco el test dentro del describe, por costumbre. Es opcional pero aconsejable

```
describe('Pruebas en el archivo demo.test.js', ()=>{

test('probando, probando...deben ser iguales los strings', ()=>{
    //punto 1, iniciar

    const mensa = "Hola Mundo!"
```

```
    //punto2, aplicar estimulo

    const mensa2: `Hola mundo!`

    //punto3

    expect(mensa).toBe(mensa2);

 })

})
```

Pruebas

- Archivo a testear 02
- Para poder testear la función, es necesario exportarla usando export
- Archivo 02:

```
const nombre = "Fer"
const apell = 'Kandinski'

const nombreCompleto = `${nombre} ${apellido}`

export function getSaludo(nombre){
    return 'Hola '+ nombre}
```

- test:
    - Guardo el getSaludo en una constante para poder manejarlo en el expect
    - Para poder usar las ayudas importar jest-dom

```
import '@testing-library/jest-dom'
import {getSaludo} from '../../02-template-strings';

describe('test de pruebas en el archivo 02', ()=>{

    test('probando que getSaludo retorna Hola + nombre', ()=>{

         const nombre: 'Fer'
         const saludo = getSaludo(nombre)

        //console.log(saludo)

        expect(saludo).toBe('Hola '+ nombre)


    })
```

```
})
```

- Usando la tecla p al final del test y seleccionando el archivo en cuestión ejecuta solo ese archivo.
- Ahora pongamos que añado Carlos como valor por defecto en el archivo origen como nombre y quiero que -getSaludo debe de retornar hola Carlos si no hay argumento nombre

```
import '@testing-library/jest-dom'
import {getSaludo} from '../../02-template-strings';

describe('test de pruebas en el archivo 02', ()=>{

    test('probando que getSaludo retorna Hola + nombre', ()=>{

         const nombre: 'Fer'
         const saludo = getSaludo(nombre)

        //console.log(saludo)

        expect(saludo).toBe('Hola '+ nombre)
    }),
    test('getSaludo debe de retornar hola Carlos si no hay argumento nombre'()=>{
        const saludo = getSaludo()
        expect(saludo).toBe('Hola Carlos')

    })



})
```

toEqual

- Archivo a testear 05

```
export const getUser = () =>({
    uid: 'ABC123'
    username: 'SuperStar'
})

export const getUsuarioActive = (nombre) => ({
    uid: 'ABC567'
    username: nombre
})
```

- archivo test:

- Importar getUSer y la libreria jest-dom
- Exportar la dfunción en el archivo origen con export

```
import '@testing-library/jest-dom'
import {getUser, getUsuarioActive} from '.....'

describe('Test en 05 funciones', ()=>{
    test('getUser debe de retornar un objeto', ()=>{

        const userTest={
            uid: 'ABC123'
            username: 'SuperStar'
        }
        const user = getUser()
        //console.log(user)

       // expect( user ).toBe(userTest)

       expect( user ).toEqual(userTest)
    })
    test('geUsuarioActive debe de retornar un objeto', ()=>{

        const nombre = 'Juan'
        const user = getUsuarioActive(nombre)

        expect(user).toEqual){
            uid: 'ABC567'
            username: nombre
        }
    })
})
```

El toBe en este caso dará error. Si tu comparas dos objetos vacíos como iguales {} === {} da false, porque apuntan a lugares distintos de memoria.

- Hay que usar el toEqual para comparar dos objetos

Pruebas en el 07

- Archivo a testear:

```
export const retornaArreglo = ()=>{
    return ['ABC', 123]
}

const [letras, numeros] = retornaArreglo();
```

- test, hacer las importaciones pertinentes

```
    describe('pruebas a 07', ()=>{
        test('Debe de retornar un string y un numero', ()=>{
                //primero preparo el ambiente, el sujeto a testear
               const arr =  retornaArreglo;
               expect(arr).toEqual (['ABC', 123])

        })
    })
```

- Podría desestructurar arr con [letras, numeros] y usar dos expects más concretos

```
expect( letras ).toBe('ABC');
expect( type of letras ).toBe('string')

expect( numeros ).toBe(123);
expect( type of numeros ).toBe('number')
```

Pruebas en 08

- Hay una dependencia de data.heroes que no está, hay que volver a crear el archivo en data/heroes.js con el arreglo de objetos de superhéroes
- Importar heroes e importar getHeroeById y getHeroesByOwner

```
export const getHeroeById = (id) => heroes.find((heroe)=> heroe.id === heroe);

export const getHeroesByOwner = (owner) => heroes.find((heroe)=>heroe.owner ===
owner);
```

```
test('debe de retornar un heroe por id', ()=>{

    const id = 1;
    const heroe = getHeroeById(id)
    const heroeData= heroes.find( h => h.id === id) //esto arroja la data de heroe
por id

    expect( heroe ).toEqual(heroeData)

})
```

- Podría mandar un id que no existiera, debe de retornar undefined

```
    const id = 10
    const heroe = getHeroeById(id)
    const heroeData= heroes.find( h => h.id === id) //esto arroja la data de heroe
por id

    expect( heroe ).toBe(undefined)  //al ser un primitivo se puede usar el toBe
```

```
test('debe de retornar un arreglo con los héroes de DC')
    const owner = DC
    const heroes = getHeroesByOwner(owner)

    const heroesData = heroes.filter((heroe)=>heroe.owner === owner)

    expect(heroes).toEqual(heroes.data
    )
```

- Puedo evaluar si es correcto sabiendo que solo hay dos heroes de marvel con el .length

```
expect(heores.length).toBe(2)
```

Pruebas con tareas asíncronas

- Archivo 09

```
import {getHeroeById} from '.....'
export const getHeroeByIdAsync = (id) =>{

    return new Promise((resolve, reject)=>{

        setTimeout(()=>{
            const p1 = getHeroeById( id)
        if (p1){
            resolve(p1)
        }else{
            reject( 'No se encontró el héroe')
        }
       }, 200)
    })
}
```

- test:
- getHeroe.. devuelve una promesa, espero que me devuelva el heroe

```
test('getHeroe debe de retornar un heroe', ()=>{

    const id = 2;
    heroe = getHeroeByIdAsync(id)
    .then(heroe=>{

        expect(true).toBe(false)
    })
})
```

- Por increible que parezca , esta prueba pasa porque así se ejecutan de manera síncrona. No se ejecuta el código del expect
- Se le puede mandar un argumento al callback, en este caso done
    - Esto le dirá a mi test suite cuando debe de terminar la prueba

```
test('getHeroe debe de retornar un heroe', (done)=>{

    const id = 2;
    heroe = getHeroeByIdAsync(id)
    .then(heroe=>{

        expect(heroe).toBe(heroes.id);
        done();
    })
})
```

- Si el heroe no existe mandaría el reject de 'No se encontró el héroe'
- Para manejar el error, cuando hay tareas asíncronas, hay que mandar el done e invocarlo

```
test('debe de retornar un error si heroe no existe', (done)=>{
    const id= 132;
    getHeroeByIdAsync(id).
    catch(error =>{
        expect(error).toEqual('No se encontró el héroe')
        done()
    })

})
```

Prueba con async-await

NOTA: faltaba la desestructurarión de la url en el archivo original

```
    const getImagen = async()=>{

        try{

            const apiKey = '1234abc';
            const resp = await fetch(`http:endpoint.random?apikey=${ apiKey}`);
            const {data} = await resp.json();

            const {url} = data.images.original
            //el mismo código para crear la imagen en el html

            const img = document.createElement('img')
                img.src= url;
                document.body.append( img )

    }catch(error){
        console.error(error)
    }
}
getImagen()
```

- archivo de test: importar la función, la librería jest-dom para ver las ayudas
- Si lleva async es que retorna una promesa

```
test('debe de retornar la url de la imagen', async ()=>{

    const url = await getImagen(); //este url es una promesa
    expect(url.includes('https://')).toBe(true)
})
```

useState

- Manejo el state de mi componente con *useState*. Cuando hay mucha data o states complejos se usa el *useReducer*
- Su snippet es así

```
const [state, setState] = useState(initialState)
```

- State es el estado en si, y SetState la función que voy a usar para cambiar mi estado
- Normalmente usaré la desestructuración y el spread para esta tarea, ya que -Cuando uso el setstate le caigo encima al state anterior, con lo cual uso el spread para importar el estado anterior, seguido del nuevo valor a añadir, si ese es el caso.
- Siempre es necesario hacer la importación

```
import {useState} from 'react'
```

Ej:

```
const [counter, setCounter] =useState ({
  counter1: 10,
  counter2: 20
})
console.log(counter)    //{counter1:10, counter2:20}

//Desestructuración

const [{counter1, counter2}] = useState ({
  counter1:10,
  counter2:20
})
```

- Por ejemplo, si necesito alterar mi state con un botón, extraigo con desestructuración del state y uso el spread con el setState para no sobreescribir el estado anterior y mandar una copia de todos los valores anteriores

```
const [state ,setState] = useState({
  counter1: 10,
  xounter2: 20,
  counter3: 30
})
```

```
const [counter1,counter2] = state;

<button onClick={()=>{
  setState({
    ...state,
    counter1: counter +1
  })
}}>
```

useEffect

- Se usa para ejecutar algún efecto secundario cuando algo cambie en el state; monitorear según dependencias los cambios en el state para ejecutar algo
- Su snippet es algo extraño, por su ciclo de vida.
    - El return del useEffect devuelve algo cuando el componente ya está montado.
- Usualmente recibe un callback como primer parámetro y un arreglo de depoendencias como segundo.
- Se trabajan de manera individual
- Notar que el arreglo vacío de las dependencias hace que solo se ejecute una vez
- Siempre se ejecuta minimo 1 vez, con la primera renderización
- Una forma de su uso habitual es para prevenir el refresh del navegador y peticiones innecesarias

```
useEffect(()=>{        //sólo se ejecuta la primera vez
  e.preventDefault()
}, [])  //con este arreglo vacío de las dependencias
```

- Muy usado en formularios
- El name es importante que coincida, ya que me servirá para visualizar lo que escribo en pantalla junto con el handleInputChange

```
  <form>
    <input
    type="text"
    name="name"
    value= {name}
    onChange={handleInputChange}
  >

</form>
```

- Defino el handleInputChange. Puedo pasar el evento y hacer un console.log para ver por dónde va. Con target.name y target.value obtengo el elemento que cambió y el valor del input

```
const handleInputChange =(e)=>{
  console.log(e.target.name)
```

```
  console.log(e.target.value)
}
```

- Desestructuro el target del e.target y uso el spread para el state.
- Computo, quiero que el nombre de esta propiedad sea lo que venga del objeto

```
const handleInputChange = ({target}) =>{
  setState({
    ...state,
    [target.name] : target.value
  })
}
```

- Es importante el apartado name = name del input. Name es el campo, value es lo que escribo en él.
- Con el arreglo de dependencias vacío, evito que a cada cambio se realice una petición/ejecución.
- Si lo que quiero es controlar el estado del formulario, puedo ponerlo como dependencia

```
useEffect(()=>{

},[formState])
```

- Si quisiera que solo esté atento si el campo del email cambia creo un nuevo efecto

```
useEffect(()=>{

//lógica

}, [email])
```

- Se usa para escuchar cambios específicos de algun componente de la aplicación

UseEffect unmount-Cleanup

No se pueden tener hooks que se rendericen de manera condicional

- Deben estar lo más arriba posible del componente
- La función que se ejecuta en el return del useEffect se efectúa cuando el componente/state esta desmontado
- Puedo renderizar de manera condicional de esta forma
- Si name existe muestra el mensaje

```
{ name === '123' && <Message />}
```

- Ahora cuando escribo 123, y solo cuando es 123 en el input se muestra el mensaje.
- El snippet es:

```
useEffect(()=>{
  effect //componente montado
  return{
    clean //componente desmontado
  }
}, [input]) //dependencias
```

- Hay un problema que solventa el cleaner del useEffect, y es que si monto un listener, aunque se desmonte el componente sigue el listener montado .Si abro varios y todos hacen peticiones vana consumir la cpu o agotar el plan de datos del usuario.

```
useEffect(()=>{

  const mouseMove = (e) =>{
    const coors={x: e.x, y: e.y}
    console.log(coors)
  }

  window.addEventListener('mousemove',mouseMove)

  return () =>{
    window.removeEventListener('mousemove')
  }

}, [])
```

- Puedo extraer las coords e imprimirlas en pantalla

```
const [coords, setCoords] = useState({x:0, y:0})

const {x,y}= coords;

useEffect(()=>{

  const mouseMove = (e) =>{
    const coords={x: e.x, y: e.y}
    setCoords(coords)
```

```
  }

  window.addEventListener('mousemove',mouseMove)

  return () =>{
    window.removeEventListener('mousemove')
  }

}, [])

return{
  <p> x : {x}, y: {y}</p>
}
```

Custom Hook useFetch

```
export const useForm = (initialState = {}) =>{

    const [values, setValues] = useState(initialState)


    const handleInputChange = ({target}) =>{
      setValues({
        ...values,
        [target.name]: target.value
      })
    }

    return [values, handleInputChange]

}
```

```
const [formValues, handleInputChange] =useForm)({
  name:''.
  email:''
  password:''
})

const [name, email, password] = formValues;

return(
  (...)formulario(...)
)
```

useFetch CustomHook

- Para crear un customHook para hacer peticiones:
- Data en null pq no tengo la data todavía
- El loading en true pq estará cargando
- Error en null porque no lo se todavía, ya lo manejaré

```
export const useFetch = (url)=>{

  const [state, setstate] =useState({data: null, loading: true, error: null})

}
```

- Una vez cambie o reciba una url voy a disparar un efecto
- Llamo al setState con el loading en false, pq si llegué hasta ahi ya no esta cargando
- El error en null, porque en principio salió bien
- Y la data es la data

```
export const useFetch = (url)=>{

  const [state, setstate] =useState({data: null, loading: true, error: null})

 useEffect( ()=>{
    fetch(url)
   .then(resp=>resp.json())
   .then(data=>{
     setState({
       loading: false,
       error: null
       data
     })
   })
  }, [url])
  return state  //aqui simplememnte retorno el state (data null, loading: true...)
}
```

- Si hago un console.log de lo que regresa

```
const state = useFetch('https:api.com/frases_e_imagenes_bracking_bad/1')
console.log(state)
```

- Regresa un objeto con loading en false, error null y en la data un arreglo con -el quote_id -el quote -el author
    - la serie

- Yo puedo desestructurar el useFetch
- Tengo dos elementos, un loading o el párrafo con el blocquote para la frase y el autor
- Bien puedo usar un ternario para visualizarlos

```
const {loading} = useFetch('https:api.com/frases_e_imagenes_bracking_bad/1')

{
  loading
  ?
    (
      <div>Loading....</div>
    )
  :
    (
      <p> {quote}</p>
      <blockquote>{author}</blockquote>
    )
}
```

- Como en la data primero hay null y luego hay data, debo de hacer una vadilación primero.
- Dice: si existe la data, extrae la posición 0 de la data.
- entonces yo debería tener author y quote en la data

```
      const {loading, data} =
useFetch('https:api.com/frases_e_imagenes_bracking_bad/1')
  const {author, quote} = !!data && data[0] //no me queda claro. !!null === false
```

useRef

- useRef devielve un objeto que se llama current

```
const inputRef = useRef()
console.log(inputRef)
```

- Puede servir para darle seguimiento a cualquier objeto

```
const handleClick = () =>{
  inputRef.current.focus() //para seleccionar con sombreado
  console.log(inputRef) //---> tengo la referencia a todo el input
}
```

```
<input
ref={inputRef}
placeholder="Su nombre...">
```

useRef caso real

- Puedo usar el .current como referencia en el tiempo , por ejemplo durante una petición http, para que no de error
- Por ejemplo, en este caso creo el estado show que es un valor booleano
- Si show esta en true muestra el componente
- Creo un botón con la función en el propio botón
- Donde la accíon del SetShow es ejecutar el valor opuesto de show

```
const [show, setShow] = useState(false)

return(
  <>
  {show && <Componente />}

  <button
  onClick={()=>setShow(!show)}>
  </button>
  </>
)
```

- Si la petición tarda mas de lo habitual, al no traer una respuesta daría error
- Uso el use ref poniendo un isMounted.
- La idea es que mantenga la referencia cuando el componente esta vivo
- Cuando cambie los valores del isMounted no quiero mandar una renderización nuevamente.

```
const isMounted = useRef(true)

const [state, setState] = useState({data:null, loading:true, error:null})

useEffect(()=>{

}, [])
```

- Yo se que el return del useEffect es cuando se desmonta el state
- Entonces si el isMounted esta montado, yo puedo llamar al setState sin problema

```
useEffect(()=>{
  isMounted.current = false
}, [])

if(isMounted.current){
  setState({
    loading:false,
    error:null,
    data
  })
}
```

Memo

- Se usa habitualmente envolviendo todo el componente desde el igual con React.memo()
- Cuando React nota un cambio en el state, vuelve a redibujar el componente
    - Para que eso no pase innecesariamente con algun componente hijo, se puede usar el memo

```
export const Memorize =React.memo((propsMemorize)=>{

  //toda la lógica entre paréntesis
})
```

useMemo

- Por ejemplo: si tengo un componente pesado, que efectua un procedimiento muy pesado, -Si cada vez que renderizo al notar React algún cambio se dispara es un engorro
- Para evitar eso uso el useMemo
- el snippet es muy parecido al useEffect: recibe un callback y una dependencia -si algo cambia, quiero memorizar el resultado de esa función, qwue en este caso sería *el proceso pesado*

```
const memoProcesoPsado = useMemo(()=> procesoPesado(counter),
[objeto:que:hace:disparar:la:renderización:counter])
```

- Le paso como función al useMemo el proceso pesado, y la dependencia me va a decir cuando memorizar esa funcion.

useCallback

- Uno de los 2 usos principales, es cuando necesito mandar una función a un componente hijo.
- Este useCallback me va a servir como función memorizada que puedo enviar como las props
- Y React sabrá si la función no va a cambiar o no ha cambiado si la dependencia no ha cambiado

```
useCallback(()=>{
  setCounter( counter +1)
}, [setCounter]) //si pusiera el counter no me serviría pq se volvería a ejecutar
                  //pero sin el counter React me manda un error
```

- La guardo en una variable

```
const increment =useCallback(()=>{
  setCounter( counter +1)
}, [setCounter])
```

- Recordar, cuando no se tiene acceso al state, se puede usar un callback

```
const increment = useCallback (()=>{
  setCounter(counter =>counter +1)
})

return(
  <div>
  <h1> useCallback Hook {counter} </h1>
  <hr/>
  <ShowIncrement increment = {increment}>

</div>

)
```

- El componente quedaría así

```
export const showIncrement = React.memo(({increment}) =>{
  return(

    <button onClick={ ()=> {
      increment
    }}> Incrementar </button>
  )
}
```

```
)
```

- La versión memorizada es la que se está mandando como argumento y también uso el React.memo para que memorice este componente si los argumentos no cambian.
- Si quisiera mandarle un parametro al incrementar puede hacerse así

```
      const increment = useCallback ((num)=>{
        setCounter(counter =>counter +num)
      })
```

Creación del Router

- Creo el HeroesApp.js como apunto de entrada de mi aplicación
- Defino las páginas/componentes, cada una en su carpeta correspondiente dentro de components
- carpeta DC en components : DCScreen.js
- carpeta Marvel en components : MarvelScreen.js
- SearchScreen.js, LoginScreen.js
- carpeta UI: Navbar.js
- La propiedad isActive se explica más adelante, igual que NavLink y useNavigate
- Copio un navbar de bootstrap, NOTA: navbar copiado del codigo finalizado

```
import React from 'react'
import { Link, NavLink, useNavigate } from 'react-router-dom'


export const Navbar = () => {

    const navigate = useNavigate();

    const handleLogout = () => {
        navigate('/login', {
            replace: true
        });
    }

    return (
        <nav className="navbar navbar-expand-sm navbar-dark bg-dark">

            <Link
                className="navbar-brand"
                to="/"
            >
                Asociaciones
            </Link>

            <div className="navbar-collapse">
                <div className="navbar-nav">

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/marvel"
                    >
                        Marvel
```

```
                    </NavLink>

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/dc"
                    >
                        DC
                    </NavLink>

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/search"
                    >
                        Search
                    </NavLink>
                </div>
            </div>

            <div className="navbar-collapse collapse w-100 order-3 dual-collapse2
d-flex justify-content-end">
                <ul className="navbar-nav ml-auto">

                    <span className="nav-item nav-link text-info">
                        Fernando
                    </span>

                    <button
                        className="nav-item nav-link btn"
                        onClick={ handleLogout }
                    >
                        Logout
                    </button>
                </ul>
            </div>
        </nav>
    )
}
```

- Para manejar las rutas creo una carpeta llamada routes.
- Creo AppRouter.js con este código
- Routes solo puede usarse dentro del contexto. -Por ello importo y añado BrowserRouter
- Muestro el NavBar

```
import {Routes,Route, BrowserRouter} from 'react-router-dom'
import{MarvelScreen} from './MarvelScreen
import{DcScreen} from './DcScreen
import{SearchScreen} from './SearchScreen
import{Login} from './LoginScreen
import {NavBar} from './NavBar'
```

```
export AppRouter =()=>{
    return(

        <BrowserRouter>


            <NavBar />

        <Routes>

            <Route path="/" element={<MarvelScreen />}>
            <Route path="/dc" element={<DcScreen />}>
            <Route path="/search" element={<SearchScreen />}>
            <Route path="/login" element={<LoginScreen />}>

        </Routes>


        </BrowserRouter>
    )
}
```

- HeroesApp renderiza AppRouter

```
const HeroesApp () =>{

    export const HeroesApp = () =>{
        return(
            <AppRouter />
        )
    }
}
```

Colocación clase Activa

- Para usar un segundo router, uno con el login y otro con el NavBar y el logout
- En la pantalla de Login no deberí amostrarse el router
- Creo un nuevo componente llamado DashboardRoutes.js
- En el Dashboard traspoaso todos los links y el navBar menos el Login

```
import { Routes, Route } from 'react-router-dom';

import { Navbar } from '../components/ui/Navbar';

import { DcScreen } from '../components/dc/DcScreen';
import { MarvelScreen } from '../components/marvel/MarvelScreen';
```

```
import { SearchScreen } from '../components/search/SearchScreen';
import { HeroScreen } from '../components/hero/HeroScreen';


export const DashboardRoutes = () => {
    return (
        <>
            <Navbar />

            <div className="container">
                <Routes>
                    <Route path="marvel" element={<MarvelScreen />} />
                    <Route path="dc" element={<DcScreen />} />

                    <Route path="search" element={<SearchScreen />} />
                    <Route path="hero/:heroeId" element={<HeroScreen />} />

                    <Route path="/" element={<MarvelScreen />} />

                </Routes>
            </div>
        </>
    )
}
```

- Y el AppRouter así

```
import { Routes, Route, BrowserRouter } from 'react-router-dom';

import { LoginScreen } from '../components/login/LoginScreen';

import { DashboardRoutes } from './DashboardRoutes';



export const AppRouter = () => {
    return (
        <BrowserRouter>

            <Routes>

                <Route path="/login" element={<LoginScreen />} />

                <Route path="/*" element={ <DashboardRoutes />  } />

            </Routes>
        </BrowserRouter>
    )
}
```

- El asterisco signfica que todas las rutas despues del slash, lo que no es login van a ser manejadas por ahi
- Si no es Login, todas las demás van por ahi
- En el Login creo un botón
- Una vez pasado el Login yo no debería poder volver ahí con el navegador
- Importo useNavigate de react-router-dom
- Usando el useNavigate puedo indicar la ruta
- El segundo parámetro implicito es el replace. Reemplazar la vista actual en lugar de crear una nueva en la historia

```
import {useNavigate} from'react-router-dom'

export const LoginScreen = () => {
  const navigate = useNavigate


  const handleLogin = () =>{
      navigate('/marvel', {
          replace: true
      })
  }


  return (
    <div>
       <h1>LoginScreen</h1>
       <button> Login </button>
    </div>
  )
}
```

- En el NavBar hago el handleLogOut
- También con el Navigate:

```
export const Navbar=()=>{
    const navigate = useNavigate();
    const handleLogOut = () => {
        navigate('/login', {
            replace: true
        })
    }
}
```

Lista de Heroes

- Para hacer el ejercicio copio data del curso

```
export const heroes = {

    {
            'id': 'dc-robin',
            'superhero':'Robin/Nightwing',
            'publisher':'DC Comics',
            'alter_ego':'Dick Grayson',
            'first_appearance':'Detective Comics #38',
            'characters':'Dick Grayson'
        },
        {
            'id': 'dc-blue',
            'superhero':'Blue Beetle',
            'publisher':'DC Comics',
            'alter_ego':'Dan Garret',
            'first_appearance':'Mystery Men Comics #1',
            'characters':'Dan Garret, Ted Kord, Jaime Reyes'
        },
        {
            'id': 'dc-black',
            'superhero':'Black Canary',
            'publisher':'DC Comics',
            'alter_ego':'Dinah Drake',
            'first_appearance':'Flash Comics #86',
            'characters':'Dinah Drake, Dinah Lance'
        },
        {
            'id': 'marvel-spider',
            'superhero':'Spider Man',
            'publisher':'Marvel Comics',
            'alter_ego':'Peter Parker',
            'first_appearance':'Amazing Fantasy #15',
            'characters':'Peter Parker'
        },
        {
            'id': 'marvel-captain',
            'superhero':'Captain America',
            'publisher':'Marvel Comics',
            'alter_ego':'Steve Rogers',
            'first_appearance':'Captain America Comics #1',
            'characters':'Steve Rogers'
        },
        {
            'id': 'marvel-iron',
            'superhero':'Iron Man',
            'publisher':'Marvel Comics',
            'alter_ego':'Tony Stark',
```

```
            'first_appearance':'Tales of Suspense #39',
            'characters':'Tony Stark'
        }
}
```

- Creo dos archivos getHeroeById.js y getHeroesByPublisher.js en una nueva carpetya llamada selectors
- getHoeroesByPublisher

```
import {Heroes} from './data'


export const getHeroesByPublisher = (publisher) =>{



    return(
        heroes.filter((hero)=>hero.publisher === publisher)
    )
}
```

- Creo HeroList, import getHeroesByPublisher

```
import {getHeroesByPublisher} from './getHeroesByPublisher'

export const HeroList = ({publisher}) =>{

    const heroes = getHeroesByPublisher(publisher) //esto es lo que me va a
permitir retornar la lista de heroes

 return {

    <div className="card-columns">
        {heroes.map(hero =>(
            <li key={hero.id}>

            {hero.superhero}

            </li>
        ))}
        <li>

        </li>
    </div>
```

```
 }                                                    //by Publisher
}
```

- Manejo de errores

```
import {Heroes} from './data'


export const getHeroesByPublisher = (publisher) =>{

    const validPublishers = ['DC Comics', 'Marvel Comics']

    if(!validPublishers.includes(publisher)){

        throw new Error(`${publisher} is not a valid Publisher`)
    }

    return(
        heroes.filter((hero)=>hero.publisher === publisher)
    )
}
```

- MarvelScreen

```
import {HeroList} from '../HeroList'

export const MarvelScreen = () =>{
    return (
        <div>
        <h1>MarvelScreen</h1>

        <HeroList publisher ="Marvel Comics">

        </div>
    )
}
```

- Esto imprime el listado de nombres de Marvel Comics
- Hago lo mismo en DcScreen

- Creo HeroCard.js

```
export const HeroCard =(
     {  id,
        superhero,
        publisher,
```

```
        alter_ego,
        first_appearance,
        characters
    })=>{
        return (

            <div>
                <p> {superhero} - {id}</p>
            </div>
        )
    }
```

- En HeroList transformo el li en el componente HeroCard
- Le paso por las props el hero desestructurado
- Lo imprime por el HeroCard

```
import {getHeroesByPublisher} from './getHeroesByPublisher'

export const HeroList = ({publisher}) =>{

    const heroes = getHeroesByPublisher(publisher) //esto es lo que me va a
permitir retornar la lista de heroes

 return {

    <div className="row rows-cols-1">

        {heroes.map(hero =>(

            <HeroCard key={hero.id}
            {...hero}
            />
        ))}
        <li>

        </li>
    </div>



 }                                                  //by Publisher
}
```

- Inserto la imagen el HeroCard

```
export const HeroCard =(
     {  id,
        superhero,
        publisher,
```

```
        alter_ego,
        first_appearance,
        characters
    })=>{
        return (

            <div>
                <img src={`/assets/heroes/${id}}.jpg`} className="card-img" alt=
{superhero} />

            </div>
        )
    }
```

- Puedo guardar en una constante el "endpoint" fuera del return. Lo demás es CSS

```
const imagePath = `/assets/${id}`

return(

    <div>
        <img src{imagePath} className="card-img" alt={superhero} />

)
```

- Creo un enlace con más info importando Link

```
<Link to={`/hero/${id}`}> Mas info... </Link>
```

Leer Argumentos por URL

- Para especificar en la barra de navegación el segmento donde quiero que vaya es necesario declararlo en el router
- Por ejemplo, si quiero que sea lo que sea vaya después de hero, pongo /:

```
<Route path="hero/:heroId)" element={HeroScreen}>
```

- Ahora si voy a heroe/spiderman-marvel aparece HeroeScreen, en blanco, pero si voy a heroe sin nada más no hay nada
- Entonces, ahora debo poder asegurarme a recibir el id del heroe y hacer algo con él
- Para esto hay un hook: useParams

```
import {useParams} from 'react-router-dom'


export const HeroScreen = () =>{

    const params= useParams();

    console.log(params)

    return(
        <div>
            <h1>HeroScreen</h1>
        </div>
    )
}
```

- La respuesta del console.log es *: "her/marvel/spider, y heroId:"marvel-spider
- Como defini en la ruta heroeId, ahora tengo el valor ahi, cuando clico en el mas info de la card que me lleva al HeroScreen en blanco por ahora
- si yo hago un console.log de params.heroeId tengo el marvel-spiderman, la card donde cliqué
- Puedo usar la desestructuración

```
const{heroeId} = useParams;

console.log(heroeId)
```

- Ahora hay que extraer de la base de datos al heroe que corresponde a ese id
- Para ello creo el getHeroById

```
import {Heroes} from './Heroes'

export const getHeroById =(id='')=>{
    return heroes.find((hero)=>hero.id === id)
}
```

- Ahora en HeroScreen:

```
import {useParams} from 'react-router-dom'


export const HeroScreen = () =>{

    const {heroId}= useParams();

    const hero=getHeroById(heroId)
```

```
    return(
        <div>
            <h1>{hero.superhero}</h1>
        </div>
    )
}
```

- Ahora en el h1 me imprime el nombre correspondiente dandole clic a mas info
- Pero si pongo una url erronea salta error y creashea la app
- El Navigate es un componente que me ayuda a redireccionar cuando es posible hacerlo
    - Ahora si introduzco una url equivocada me lleva al home

```
import {useParams} from 'react-router-dom'


export const HeroScreen = () =>{

    const {heroId}= useParams();

    const hero=getHeroById(heroId)

    if(!hero) {
        return <Navigate to='/'>
    }

    return(
        <div>
            <h1>{hero.superhero}</h1>
        </div>
    )
}
```

- Cómo hago para mostrar la info de la card correspondiente? - usando css y los params. ej:
- Creo un botón en HeroScreen para regresar a la página anterior con el handleReturn en el onClick

```
<div>

<img src={imagePath}

alt={superhero}/>
</div>

<div>
<li className="list-group-item" <b>Alter ego: </b> {alter_ego} >
```

```
</div>
```

- Importo useNavigate.
- Con -1 vuelve a la página anterior

```
import {useParams,Navigate, useNavigate} from 'react-router-dom'
import {getHeroById} from '../getHeroById'

export const HeroScreen = () =>{

    const {heroeId} = useParams();

    const navigate = useNavigate();

    const hero= getHeroById(heroId)M

    const handleReturn = () =>{
        navigate('-1')
    }
}
```

useMemo

- Quiero memorizar hero en HeroScreen, para que solo renderice cuando haya un cambio en este y no haga mil peticiones

```
export const HeroScreen = () =>{

    const {heroeId} = useParams();

    const navigate = useNavigate();

    const hero= useMemo(()=>{
     return  getHeroById(heroId)

    }, [heroId])

    const handleReturn = () =>{
        navigate('-1')
    }
}
```

- En HeroList igual

```
import { useMemo } from 'react';

import { HeroCard } from './HeroCard';
import { getHeroesByPublisher } from '../../selectors/getHeroesByPublisher';

export const HeroList = ({ publisher }) => {

    const heroes = useMemo( () => getHeroesByPublisher( publisher ), [ publisher ]
);

    return (
        <div className="row rows-cols-1 row-cols-md-3 g-3 animate__animated
animate__fadeIn">
            {
                heroes.map( hero => (
                    <HeroCard
                        key={ hero.id }
                        { ...hero }
                    />
                ))
            }
        </div>
    )
}
```

SearchComponent

- El componente está en el DashboardRoutes "/search"
- Si yo pongo /search/:term va a ser un termino obligatorio
- Quiero que sean opcionales
- Hago, copio, su espacio en el NavBar
- Voy a usar el useForm anteriormente hecho

```
  import { useState } from 'react';


export const useForm = ( initialState = {} ) => {

    const [values, setValues] = useState(initialState);

    const reset = () => {
        setValues( initialState );
    }


    const handleInputChange = ({ target }) => {

        setValues({
            ...values,
```

```
            [ target.name ]: target.value
        });

    }

    return [ values, handleInputChange, reset ];

}
```

- Hago uso del useForm importándolo
- en la declaración guardo el searchText vacío en el state del useForm
- Lo extraigo con desestructuración para trabajar mejor
- coloco en el Onchange el handleInputChange - Este recibe el evento, el evento va al useForm, extrae el name y el value ylos coloca
- el handleSearch va a estar en el onSubmit del formulario

```
import {useForm} from '/useForm'


export const SearchScreen =()=>{

const handleSearch = () =>{

    const [formValues, handleInputChange] = useForm({
        searchText:'',
    })

    const{searchText} = formValues

}
return(

    <>
        <h1>Búsqueda</h1>
        <hr />

        <div>

            <div>

            <h4>Buscar</h4>
            <hr />

            </div>
            <form onSubmit={handleSearch}>
                <input
                tyoe="text"
                placeholder="Buscar un heroe..."
                name="searchText"
                value={searchText}
```

```
                onChange= {handleInputChange}/>

                <button
                onClick="handleSearch"
                type="submit">
                Buscar.. </button>

            </form>


        </div>


    </>
)

}
```

- handleSearch: Cuando el primer argumento es el argumento que yo quiero llamar en una función que estoy llamando, simplemente lo mando entre paréntesis. Vendría a ser esto

```
<form onSubmit={(e)=>handleSearch(e)}>
```

- Debo mandar el evento con el e.preventDefault para evitar el refresh

```
import {useForm} from '/useForm'


export const SearchScreen =()=>{

    const [formValues, handleInputChange] = useForm({
        searchText:'',
    })

    const{searchText} = formValues

    const handleSearch = (e) =>{

   e.preventDefault();

    console.log(searchText)

}
return(

    <>
        <h1>Búsqueda</h1>
```

```
        <hr />

        <div>

            <div>

            <h4>Buscar</h4>
            <hr />

            </div>
            <form onSubmit={handleSearch}>
                <input
                tyoe="text"
                placeholder="Buscar un heroe..."
                name="searchText"
                value={searchText}
                onChange= {handleInputChange}/>

                <button
                onClick="handleSearch"
                type="submit">
                Buscar.. </button>

            </form>


        </div>


    </>
)

}
```

Mostrar Listado Heroes

- Creo el archivo getHeroesByName

```
import {Heroes} from './heroes'

export const getHeroesByName = (name = '') =>{

    name = name.toLowerCase()

    return heroes.filter(hero=> hero.superhero.toLowerCase().includes(name))


}
```

- En el map.filter regreso el HeroCard, que espera todos esos valores de name, etc y al ser .map necesita un key

```
import {useForm} from '/useForm'
import {getHeroesByName} from './getHeroesByName'
import {HeroCard} from './HeroCard'


export const SearchScreen =()=>{

    const [formValues, handleInputChange] = useForm({
        searchText:'',
    })

    const{searchText} = formValues

    const heroesFiltered = getHeroesByName()

    const handleSearch = (e) =>{

   e.preventDefault();

    console.log(searchText)

}
return(

    <>
        <h1>Búsqueda</h1>
        <hr />

        <div>
            {
                heroesFiltered.map(hero=>(
                    <HeroCard key={hero.id}
                    {...hero}/>
                ))
            }
        </div>

            <div>

            <h4>Buscar</h4>
            <hr />

            </div>
            <form onSubmit={handleSearch}>
                <input
                tyoe="text"
                placeholder="Buscar un heroe..."
                name="searchText"
```

```
                value={searchText}
                onChange= {handleInputChange}/>

                <button
                onClick="handleSearch"
                type="submit">
                Buscar.. </button>

            </form>


        </div>


    </>
)

}
```

Aplicar filtro de heroes- QueryString

- Yo lo que quiero es hacer lo que hace google, poner ?q=Batman&casa=DC etc
- Construir los query parameters desde la búsqueda
- Quiero navegar a la ruta donde este el queryparameter establecido
- Para ello uso el navigate con backtics

```
const handleSearch=()=>{

  e.preventDefdault();

    navigate(`?q=${searchText}`)
}
```

- Ahora que ya puedo establecerlo en la barra de navegación, necesito poder leerlo desde esta
- Necesito el Location, para eso está el useLocation, lo importo~y lo llamo

```
export const SearchScreen =()=>{
    const navigarte =useNavigate()
    const location = useLocation()

    const [formValues, handleInputChange] = useForm({
        searchText:'',
    })

    const{searchText} = formValues

    const heroesFiltered = getHeroesByName()
```

```
    const handleSearch = (e) =>{

        e.preventDefault();

       navigate(`?q=${searchText}`)
    }
```

- Si hago un console.log el location veré cierta info relevante: hay un key, un pathname, un search, un state,...
- Instalar este paquete facilita este trabajo

```
npm install query-string
```

- Se importa. Si ahora hago un console.log de location.search obtengo en consola la string de parameter que tengo que enviar

```
const query = queryString.parse(location.search)
```

- A mi lo que me interesa es la q de esa info, puedo desestructurar y mandar un valor por defecto

```
const {q=''} = queryString.parse(location.search)
```

- Para que el parámetro de busqueda según la url aparezca en el campo de texto, en lugar de SearchText: '' con un string vacío le paso q
- Al getHeroesByName del searchScreen le paso el query

```
const heroesFiltered = getHeroesByName(q)
```

Hago una comparación en la función

```
import {Heroes} from './heroes'

export const getHeroesByName = (name = '') =>{


    if( name.length === 0){
        return []
    }
```

```
    name = name.toLowerCase()
    return heroes.filter(hero=> hero.superhero.toLowerCase().includes(name))


}
```

Mostrar Mensajes Condicionales

- Necesito mostrar algunos mensajes consdicionales. Por ejemplo cuando se busca y no se encuentra
- Por ejemplo, si query es un string vacio, entonces regreso un componente

```
return(

    <>
        <h1>Resultados</h1>
       <hr />
                {   (q===0)
                        && <div className="alert alert-info">Buscar un heroe</div>


                }

        <div>
            {
                heroesFiltered.map(hero=>(
                    <HeroCard key={hero.id}
                    {...hero}/>
                ))
            }

)

}
```

- Se puede hacer todo con un ternario

```
                {   (q===0)
                        ? <div className="alert alert-info">Buscar un heroe</div>
                        :(heroesFiltered.length ===0)
                        && <div className="alert alert-danger">No hay resultados:
{q} </div>

                }
```

- La función getHeroesByName dispara mogollón de peticiones.

- Solo debería procesarse cuando q cambia. Se pude solucionar con un useMemo
- useMemo recibe una función, un callback. El retorno de la función es lo que se memoriza

```
const heroesFiltered =useMemo(()=>getHeroesByName(q), [q])
```

Protección de rutas

- Creo una carpeta llamada auth y otra types con un archivo llamado types.js
    - Estas son las acciones que voy a acabar disparando
    - Login apunta a auth login

```
export const types = {
    login: '[auth] Login',
    logout: '[auth] Logout'
}
```

- Creo un archivo llamado authReducer.js
- Con un switch filtro que clase de acción
- Desestructuro el payload que todavía no he creado

```


export const authReducer = (state = {}, action) =>{

        switch (action.type){

            case types.login:
                return{

                  ...action.payload,
                  logged: true
                }
            case types.logout:

                return{

                    logged: out
                }


            default:
                return state;

                break;
        }

}
```

- el reducer debe de seer una función pura. -Debe resolver todo en su interior sin alteraciones externas ni interacciones con el exterior
- Creo el authContext.js en la carpeta auth

```
const authContext = createContext()
```

- Busco el punto más alto de mi aplicación, porque quiero que el context esté disponible en toda ella

Proveer estado global de la app

- Debo importar el AuthRouter y añadir el Provider

```
import {AppRouter} from './routers/AppRouter'
import {AuthContext} from './auth/AuthContext'

export const HeroesApp = () =>{


    return (

        <AuthContext.Provider>

       <AppRouter />

        </AuthContext.Provider>
    )
}
```

- Cuando uso el Provider debo aportar información, algo para compartir con todos sus hijos
- Puerdo poner funciones que vuelvan a renderizar componentes o algo simple como una variable
- El useReducer me va a permitir la implementación del contexto
    - Declaro 2 funciones el state y el dispatch
    - El state sería el state del authReducer con el logged, etc
    - No ejecuto el authReducer, mando la referencia nada más
    - Luego necesito un estado incial, {} y el init
    - El init se puede inicializar fuera de la función, es para que solo se ejecute una vez

```
import {AppRouter} from './routers/AppRouter'
import {AuthContext} from './auth/AuthContext'



export const HeroesApp = () =>{

    const [state, dispatch] =useReducer(authReducer, {}, init)

    return (

        <AuthContext.Provider>
```

```
        <AppRouter />

        </AuthContext.Provider>
    )
}
```

- Le cambio el nombre al state y le pongo user
- Ahora ya puedo mandarlo en el context

```
import {AppRouter} from './routers/AppRouter'
import {AuthContext} from './auth/AuthContext'

const init = ()=>{

    return {

        logged: true,
        name: "Migue Temporal"
    }
}

export const HeroesApp = () =>{

    const [user, dispatch] =useReducer(authReducer, {}, init)

    return (

        <AuthContext.Provider value={{
            user,
            dispatch
        }}>

        <AppRouter />

        </AuthContext.Provider>
    )
}
```

- Ahora si miro en components del navegador, en el contextProvider tengo el name y el logged
- Tengo la función para cambiar el state
- Cómo leer el nombre del state para mostrarlo en el navbar??

- Necesito acceso a mi useContext en el Navbar
- Si yo hago un console.log del context tengo el user y el dispatch
- Pongo user.name en el p ~

```
import React, {useContext} from 'react'
import { Link, NavLink, useNavigate } from 'react-router-dom'
import {AuthContext} from '../auth/AuthCOntext

export const Navbar = () => {

    const navigate = useNavigate();

    const context = useContext(AuthContext)

    const handleLogout = () => {
        navigate('/login', {
            replace: true
        });
    }

    return (
        <nav className="navbar navbar-expand-sm navbar-dark bg-dark">

            <Link
                className="navbar-brand"
                to="/"
            >
                Asociaciones
            </Link>

            <div className="navbar-collapse">
                <div className="navbar-nav">

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/marvel"
                    >
                        Marvel
                    </NavLink>

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/dc"
                    >
                        DC
                    </NavLink>

                    <NavLink
                        className={ ({ isActive }) => 'nav-item nav-link ' +
(isActive ? 'active' : '') }
                        to="/search"
                    >
                        Search
                    </NavLink>
                </div>
```

```
            </div>

            <div className="navbar-collapse collapse w-100 order-3 dual-collapse2
d-flex justify-content-end">
                <ul className="navbar-nav ml-auto">

                    <span className="nav-item nav-link text-info">
                        {user.name}
                    </span>

                    <button
                        className="nav-item nav-link btn"
                        onClick={ handleLogout }
                    >
                        Logout
                    </button>
                </ul>
            </div>
        </nav>
    )
}
```

- Ahora hay una manera de leer y establecer ese objeto mediante el reducer, init inicializa el reducer
- Pero voy a grabar el usuario en el localstorage, así sabré si está autenticado o no
- Esto puede no existir, por eso el operador OR para regresar el estado inicial en false
- Esto no es una autenticación rigurosa, es un ejemplo educativo para leccciones posteriores

```
const init = ()=>{

    return {

        return localStorage.getItem('user') || (logged:false)

    }
}

export const HeroesApp = () =>{

    const [user, dispatch] =useReducer(authReducer, {}, init)

    return (

        <AuthContext.Provider value={{
            user,
            dispatch
        }}>

       <AppRouter />

        </AuthContext.Provider>
```

```
    )
}
```

- En el local storage solo puedo grabar strings
- Por ello tomo el string y lo convierto a objeto, usando el JSON.parse

```
const init = ()=>{

    return {

        return JSON.parse(localStorage.getItem('user')) || (logged:false)

    }
}
```

Login de un usuario

- Ahora voy a crear un usuario apretando el button de LOGIN
- LoginScreen

```
import { useNavigate } from 'react-router-dom';


export const LoginScreen = () => {

    const navigate = useNavigate();

    const handleLogin = () => {
        navigate('/marvel', {
            replace: true
        });
    }


    return (
        <div className="container mt-5">
            <h1>Login</h1>
            <hr />

            <button
                className="btn btn-primary"
                onClick={ handleLogin }
                >
                    Login
            </button>
```

```
        </div>
    )
}
```

```
    - En el handleLogin esta el intringulis
```

- En el action hay que leer de los types /hay que importarlos / y el context del authReducer
- Tengo el types centralizado pq si pusiera un string con '[]auth Login' luego la conversión del String sería manual
- El payload debe manadar el name, pq del log ya se encarga el reducer. Si le mando el payload establece el log por mi
- Debo obtener también el dispatch para poder ejecutarla
- Como ahora estoy autenticado puedo regresar el navigate

```
const handleLogin = () =>{
    const navigate = useNavigate

    //const context = useContext(AuthContext)

    const {dispatch} = useContext(AuthContext)


    const action = {
        type: types.login
        payload: {name: 'Migue'}
    }

    dispatch(action)



     navigate('/login', {
        replace: true
    });



}
```

- Si recargo el navegador, pierdo la info, el nombre desaparece del navbar---
- Porqué se pierde esa info?
    - Por defecto todo está en memoria
    - Lo que está en el state, el authReducer, todo está en memoria
- En algun lugar tengo que grabar el localStorage.getItem

- Puedo usar un useEffect pendiente del usuario
- Como en el localStorage solo puedo grabar Strings, uso el stringify

```
import {AppRouter} from './routers/AppRouter'
import {AuthContext} from './auth/AuthContext'
import{useEffect} from 'react'



const init = ()=>{

    return {

        logged: true,
        name: "Migue Temporal"
    }
}

export const HeroesApp = () =>{

    const [user, dispatch] =useReducer(authReducer, {}, init)

    useEffect(()=> {

        if( !user) return;
        localStorage.setItem('user', JSON.stringify(user))
    })

    return (

        <AuthContext.Provider value={{
            user,
            dispatch
        }}>

       <AppRouter />

        </AuthContext.Provider>
    )
}
```

- Cuando la app es recargada, ahora se mantiene el nombre en el navbar PORQUE:
  - Se vuelve a ejecutar el código de HeroesApp
  - Pasa por el useReducer, este manda a llamar ella función del init, pasa por leer el localStorage
  - Como lo encuentra, lo parsea y regresa el objeto que es mi nuevo state

Ahora hay otro inconveniente: pese a estar logeado sigo pudiendo ver la pantalla del Login

Logout del usuario

- Para el logout lo que hay que hacer es mandar a llamar una acción de tipo logout
- Luego voy a automatizar con rutas privadas el que no se pueda acceder sin el login, pero ahora lo haré manual
- Desestructuro el useContext y mando el dispatch directamente , sin crear el action en una const aparte

```
export const Navbar = () => {

    const navigate = useNavigate();

    const {user, dispatch} = useContext(AuthContext)

    const handleLogout = () => {

        dispatch({ type: types.logout })

        navigate('/login', {
            replace: true
        });
    }

    return (
```

- en redux hay funciones para esto, pero eso más adelante
- Sigue estano el logged en false en consola cuando me voy fuera con el logout
- Porqué sucede? Porque en el transcurso renderizar de nuevo el useReducer, llama al init
- En algun punto este tiene el valor de null, por eso se queda el logged: false
- Este es el objeto, el estado inicial
- El useEffect esta pendiente de usuario y siempre se dispara la primera vez.
- Cuando cambia el usuario ya no va a ser nulo, y como no es nulo va a grabarse en el localStorage
- Y que valor tiene si el usuario no existe =? va a tener el valor del logged: false
- Este comportamiento es lo que busco en este caso

```
const init = () =>{
    return JSON.parse( localStorage.getItem('user')) || logged: false
}


export const HeroesApp = () =>{

    const [user, dispatch] =useReducer(authReducer, {}, init)

    useEffect(()=> {

        if( !user) return;
        localStorage.setItem('user', JSON.stringify(user))
    })
```

Rutas Privadas

- En este caso, el DashboardRoutes solo debería ser visible si el user esta autenticado
- Si voy al AppRouter, el Login es publica pero la otra es privada
- Pongo el PrivateRoute que no existe, lo voy a crear y renderizo dentro el Dashboard
- despues de este path /* va a renderizar el componente DashboardRoutes

```
export const AppRouter = () => {
    return (
        <BrowserRouter>

            <Routes>

                <Route path="/login" element={<LoginScreen />} />

                <Route path="/*" element={

                    <PrivateRoute>

                        <DashboardRoutes>

                    <PrivateRoute>


                    }
                />
                 //////////////<Route path="/*" element={ <DashboardRoutes />  }
/>//////YA NO VA

            </Routes>
        </BrowserRouter>
    )
}
```

- En routers creo el functional component PrivateRoutes
- Si pongo un hola mundo, al ingresar en /marvel veo el hola mundo
- Eso significa que todo lo que ponga ahi es lo que se verá después de /, además, puedo usar todo lo aprendido, useEffect, useContext, etc

```
import React from 'react'

export const PrivateRouter = () => {

  return (
```

```
    <div>Hola mundo</div>
  )
}
```

- Llamo al context, desestructuro el user
- Ahora debo regresar un componente de manera condicional
- Si está loggeado qué voy a renderizar? Los hijos de PrivateRouter!
- Los voy a recibir en las props
- En caso de no estar autenticado puedo renderizar algo

```
import {useContext} from 'react-router-dom'

export const PrivateRoute = ({children}) => {
  const {user} = useContext(AuthContext)

      return user.logged
      ? children
      : <p>No está autenticado</p>
}
```

- Pero lo interesante es sacar de esa pantalla al usuario
- Para ello uso Navigate, lo importo

```
const {user} = useContext(AuthContext)

      return user.logged
      ? children
      : <Navigate to="/login" />
```

Rutas públicas

- Creo en routers el componente PublicRoute
- Ahora solo usuarios no autenticados pueden ver el Login

```
import {useContext} from 'react-router-dom'
import {Navigate} from 'react-router-dom



export const PublicRoute = ({children}) => {

  const {user} = useContext(AuthContext)


      return user.logged

      ? <Navigate to="/marvel"

      : children

}
```

```
export const AppRouter = () => {
    return (
        <BrowserRouter>

            <Routes>

                <Route path="/login" element={

                    <PublicRoute>

                         <LoginScreen />

                    </PublicRoute>} />

                <Route path="/*" element={

                    <PrivateRoute>

                        <DashboardRoutes>

                    <PrivateRoute>


                    }
                />
                 /////////////<Route path="/*" element={ <DashboardRoutes />  }
/>//////YA NO VA

            </Routes>
        </BrowserRouter>
    )
}
```

Recordar la última página

Reducer

- Es una función común y corriente
- No puede ser asíncrona
- Debe de ser pura. Todo lo que realice tiene que resolverse internamente
- No debe de llamar al LocalStorage o SessionStorage
- Debe de retornar siempre un nuevo estado; ni errores ni otras cosas
- Solo recibe dos argumentos: el estado inicial y la acción
- Para modicificar el state solo debe de haber una acción

```
const algoReducer = ()=>{
    //esto puede ser un reducer, una función común
}
```

Un ejemplo:

```
const initialTodos = [{
    id:1,
    todo: 'Comprar té'
    done: false
}]

const todoReducer = ( state = initialTodos, action) =>{
    return state
}

let todos = todoReducer;
console.log(todos)
```

- Este console.log mostraría el objeto initialTodos con length:1
- La idea es tener controlado en un solo lugar toda la lógica que modifica mi state

---

Ciclo

- Tengo un State inicial, le pasa a la página para mostrarlos en pantalla
- El usuario que necesita agregar un nuevo todo lo ingresa, - pero la página no modifica directamente el state - La página va a crear un aacción para añadir un nuevo elemento - Esa acción es lo que le voy a mandar al Reducer - reducer tiene todo el mapa de las acciones que puede realizar - Borrar, Agregar o modificar - No debe disparar efectos secundarios

---

1- Establezco el estado inicial. 2- Declaro el reducer, siempre tiene 2 argumentos - el state y la accion 3- Si quiero agregar un nuevo objeto, como lo hago? 4- Creo un nuevo ToDo 5- Creo una acción 6- Le añado un

type, que servirá para clasificar qué tiopo de acción es 7- El payload, es un standard llamar así a los argumentos que se le quiere mandar a la acción

```
const newTodo = {
    id:2,
    todo: 'Comprar café'
    done: false
}

const action ={
    type: 'agregar',
    payload: newTodo
}

todos = todoReducer ( todos, action)
```

- Debo meter la lógica al todoReducer para procesar la acción
- El interrogante dice: si hay un action lee el type, si no no haga nada

```
const todoReducer = (state= initialState, action) =>{
    if(action?.type === 'agregar'){
        return [...state, action.payload]
    }
}

//ejemplo educativo, e un caso real se hace distinto
```

Uso del switch y el dispatch con el Reducer

- Se puede pensar en el useReducer como el useState.
- el snippet dice así

```
const [state, dispatch] = useRecuder(reducer, initialState, init)
```

- Declaro un initialState fuera del componente.
- En un archivo aparte hago el todoReducer

```
const initialState =[{
    id:new Date().getTime();
    desc: 'aprender React'
    done: false
```

```
//el switch siempre se puede utilizar si se puede comparar con un ===

export const todoReducer =(state=[], action) =>{ //pongo un arreglo vacío por
defecto
                                                 //para que no de error
        switch(action.type){
           // case 'add':



            default:
                return state;
        }
}
```

- En el archivo padre llamo al todoReducer y lo importo
- La desestructuración de objetos es posicional
    - Yo puedo renombrar el state todos

```
//const [state] = useReducer(todoReducer, initialState)
const [todos] = useReducer(todoReducer, initialState)
console.log(todos)
```

- En el archivo padre hago carpintería...

```
return{
    <div>
        <h1>TodoApp ({todos.length})</h1>
        <ul>
            {todos.map((todo, id)=>{
                <li
                key= {todo.id}>

                {i+ 1} - {todo.desc}</li>
            })}
        </ul>
    </div>
}
```

```
const handleSubmit = (e) =>{
    e.preventDefault();
```

```
}


<form onSubmit={handleSubmit}>

<input
type="text"
name="description" />

<button
type="submit"> Agregar TODO </button>

</form>
```

- Como agregar un Todo. Tengo que hacer que la info de agregar pase por el reducer

```
const handleSubmit = (e) =>{
    e.preventDefault();

   const newTodo ={
        id:new Date().getTime(),
        desc: 'Nueva Tarea',
        done: false

        }

    const action = {
        type: 'add',
        payload: newTodo
    }
```

- Uso el spread para añadir todos los todos, y añado el nuevo

```
export const todoReducer =(state=[], action) =>{ //pongo un arreglo vacío por
defecto
                                            //para que no de error
        switch(action.type){
            case 'add':
                    return[ ...state, action.payload]


            default:
                return state;
        }
}
```

- Ahora, cómo hago para mandar esta acción?
- El segundo argumento del useReducer es el dispatch
- el dispatch es a quien tengo que mandarle la acción

```
//const [state, dispatch] = useRecuder(reducer, initialState, init)

const [todos, dispatch] = useReducer(todoReducer, initialState)

const action = {
        type: 'add',
        payload: newTodo
    }
dispatch( action )
```

- Este es el proceso, mecánico:
- Crear las acciones y mandarlas al Reducer
- el Reducer las ejecuta y regresa un nuevo estado
- el useReducer se encarga de redibujar lo que haya que redibujar

---

- Para borrar, lo ideal es primero ir al reducer para especificar la tarea
- Uso el .filter para regresar un nuevo arreglo con aquellos elementos que cumplan una condición
- En este caso dice: regresame todos aquellos de id distinto al que te mando en el payload

```
export const todoReducer =(state=[], action) =>{ //pongo un arreglo vacío por
defecto
                                            //para que no de error
        switch(action.type){
            case 'add':
                    return[ ...state, action.payload];
            case 'delete':
                return state.filter(todo => todo.id 1== action.payload);


            default:
                return state;
        }
}
```

- Para hacer un delete es el mismo proceso que add
- Primero asegurarse de recibir el todoId
- Crear la acción
- Hacer el dispatch

Como necesito un argumento, hago función de flecha

```
<button
onCLick = {()=> handleDelete(todo.id)}> Borrar</button>
```

```
const handleDelete = (todoId) =>{
    //console.log(todoId)
    const action = {
        type: 'delete'
        payload = todoId
    }

    dispatch(action)


}
```

Context

- Para compartir data de cualquier tipo enter compoenntes está el context
- Está a un nivel superior. Todos aquellos componentes que esten dentro del contexto podrán acceder.
- En una página de login, tengo el user y una manera de establecer al user, setUser
- Login va a llamar al setUser para establecer el usuario.
- El homepage va a leer ese context para obtener los ultimos valores actualizados de ese usuario

---

El context es un contenedor de info que está a un nivel superior, que le va a permitir a su hijos leer y ejecutar métodos.

Introduccion al router

- El router sirve para hacer una aplicación de una sola página.
- Yo no quiero que al cambiar de pág genere una nueva petición
- Una app puede tener más de un router
- Creo un componente llamado AppRouter.js
- Añado dos rutas de dos elementos creados anteriormente

```
import {Routes, Route, BrowserRouter} from 'react-router-dom'
import {Home} from '../components/home'
import {About} from '../components/about'
import {Login} from '../components/login'

export const AppRouter = () => {
  return (
      <BrowserRouter>

        <Routes>


            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/login" element={<Login />} />

        </Routes>
    <BrowserRouter>
  )
}
```

- Uso el Link en lugar del anchor tag

```
import React from 'react'
import {Link} from 'react-router-dom'
```

```
export const NavBar = () =>{

    return{
        <div>
            <ul>
                <li>
                <Link to= "/"> Home </Link>
                </li>
                <li>
                <Link to= "/about"> About </Link>
                </li>
                <li>
                <Link to ="/login"> Login </Link>
                </li>
            </ul>

        </div>
    }
}
```

- Se puede usar redireccionamiento por si no encuentra ninguna de las opciones

```
    <Route path="/login" element={<Login />} />

    <Redirect to="/">
```

- la única diferencia con NavLink es que NavLink sirve para jugar con las clases CSS
- Copio un navbar de bootstrap
- Añado bootsrap al html en public
- La añado al AppRouter
- Voy

```
    export const AppRouter = () => {
      return (
          <BrowserRouter>
            <NavBar />
            <Routes>


                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
                <Route path="/login" element={<Login />} />
```

```
            </Routes>
        <BrowserRouter>
      )
    }
```

- Creo un userContext.js
- importo createContext
- createContext genera un contexto, lo almaceno en UserContext
- este UserContext puede contener elementos dentro, es un "higher order component"

```
import {createContext} from 'react'

export const UserContext = createContext(null)
```

- Si yo voy al main, y enmarco el AppRouter con el userContext
- Necesito el .Provider. Sirve para proveer la información
- Lo que quiero compartir se hace mediante la propiedad value

```
export const MainApp = () =>{

    const user = {
        id: 1234,
        name: "Fer",
        email: "sdldj@gmail.com"
    }


    return(
        <UserContext.Provider value={user}>

        <AppRouter />

        </UserContext.Provider>
    )
}
```

- Si miro en la consola los componentes veo que ha cambiado
- Puedo ver que hay una propiedad value con el objeto
- Si ahora necesito acceder al user desde el home por ejemplo?

```
export const HomeScreen = ()=>{
    const userContext = useContext(UserContext)
```

```
    return(
        <div>Home Screen</div>
    )
}
```

- Uso el UserContext que definí. Esto le dirá al useContext que use la primera instancia que definí en este arbol de componentes de tipo UserContext. Valor en el que guardé el contexto
- Ahora el userContext es el user que definí previamente que pasé alUserContext.provider como value
- Lo que ponga en el value va a ser accesible por aquellos elementos que estén dentro del contexto, en este caso UserContext

---

useContext

- Creo un useState
- Ahora el valor de user es un objeto vacío, lo puedo ver en components.

```
export const MainApp = () =>{

    const [user, setUser] = useState({})

    return(
        <UserContext.Provider value={user}>

        <AppRouter />

        </UserContext.Provider>
    )
}
```

- Cómo hago para enviar más arghumentos al value para compartir?
- Puedo crearlo directamente como un objeto en el propio value

```
    return(
        <UserContext.Provider value={{
            user,
            setUser,
        }}>

        <AppRouter />

        </UserContext.Provider>
    )
```

- Ahora tengo en values el user, el setUser, y el objeto vacío
- Si voy al home a hacer un console.log del userContext tengo el user y el setuser

- Uso la desestructuración para extraerlo
- Si lo imprimo tengo un objeto vacio

```
export const HomeScreen = ()=>{
    //const userContext = useContext(UserContext)
   const [user] = useContext(UserContext)
   return(
        <div>Home Screen</div>
    )
```

- componente Login:
- Quiero aplicar el setUser con el botón del componente Login

```


export const LoginScreen = ()=>{


    return(
        <h1>Login</h1>
        <button></button>
    )
}
```

```
export const LoginScreen = ()=>{

    const [setUser] = useContext(UserContext)

    return(
        <h1>Login</h1>
        <button onClick={()=>setUser({
            id: 123,
            name: "Yo mismo"
        })}></button>
    )
}
```

- Si le doy click al botón y miro en consola, ya no tengo un objeto vacío
    - sino el 123 y Yo mismo

Heroes App

- Borro todos los componentes que no voy a usar de la app.
- Borro lo innecesario del main hasta dejar solo el ReactDOM y las 2 importaciones
- Linkeo el bootstrap en el html de la carpeta public

- Creo la carpeta assets en src con las imágenes descargadas
- Creo un archivo como punto de entrada para renderizar en el index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import {HeroesApp} from './HeroesApp'


ReactDOM.render(
    <HeroesApp />,
    document.getElementById('root')
)
```

Creando un primer router

- Lo primero es definir las páginas a las que voy a navegar
- Esas páginas no son más que componentes , igual que lo demás
- Creo la carpeta components, con carpetas para DC, Marvel,Login,Search..
- En cada archivo monto un cascarón de componente
- MarvelScreen, DCScreen, LoginScreen, SearchScreen
- Tengo un navbar creado por bootstrap

Para el manejo de las rutas me creo un directorio llamado routes

```
import {Routes, Route} from 'react-router-dom'
import {MarvelScreen} from './MarvelScreen'
import {DcScreen} from './DcScreen'
import {Login} from './LoginScreen'

export const AppRouter = ()=>{
    return(
        <div>
            <h1>Welcome to react router</h1>
            <Routes>
            <Route path="/" element ={<MarvelScreen />}>
            <Route path="/marvel" element ={<MarvelScreen />}>
            <Route path="/search" element ={<SearchScreen />}>
            <Route path="/dc" element ={<DcScreen />}>
            <Route path="/login" element ={<LoginScreen />}>
        </div>
    )
}
```

Renderizo este AppRouter en mi HeroesApp

```
import {AppRouter} from './AppRouter'

export const HeroesApp = ()=>{
    return(
        <AppRouter />
    )
}
```

- Error!: useRoutes solo puede ser usado en un contexto de Router component
- En el punto mas alto, hay que envolver la app con BrowserRouter

```
import {Routes, Route, BrowserRouter} from 'react-router-dom'
import {MarvelScreen} from './MarvelScreen'
import {DcScreen} from './DcScreen'
import {Login} from './LoginScreen'
import {NavBar} from './NavBar'


export const AppRouter = ()=>{
    return(
        <BrowserRouter>
            <NavBar />

            <Routes>

            <Route path="/" element ={<MarvelScreen />}>
            <Route path="/marvel" element ={<MarvelScreen />}>
            <Route path="/search" element ={<SearchScreen />}>
            <Route path="/dc" element ={<DcScreen />}>
            <Route path="/login" element ={<LoginScreen />}>

            </Routes>

        </BrowserRouter>
    )
}
```

- ActiveClassName ya no existe. Esta coloca la clase activa cuando coincide la ruta con el link de enlace
- Creo un botón de Logout al final del NavBar
- Creo un handleLogout y lo pongo en el onClick del button
- Manejo el isActive que viene implícito en las props del NavLink
- Puedo transformar el className en una función de flecha, y desestructurar isActive de las props
- Hago un ternario diciendo si es active añade active si no, string vacío , no añadas active

- Si está en Marvel no esta en DC y viceversa
- Ahora necesito un segundo router donde este el usuario sin login, sin la navbar, todo distinto

```
import React from 'react'
import { Link, NavLink, useNavigate } from 'react-router-dom'


    export const Navbar = () => {

        const handleLogOut =()=>{
            navigate('/login', {replace: true})
        }
        const navigate = useNavigate();
        return (
            <nav className="navbar navbar-expand-sm navbar-dark bg-dark">

                <Link
                    className="navbar-brand"
                    to="/"
                >
                    Asociaciones
                </Link>

                <div className="navbar-collapse">
                    <div className="navbar-nav">

                <NavLink
                 className={({isActive}) =>"nav-item nav-link" + (isActive?
'active': '')}
                 to="/marvel">
                           Marvel
               </NavLink>

                 <NavLink
                  className={({isActive}) =>"nav-item nav-link" + (isActive?
'active': '')} to="/dc">
                           DC
                       </NavLink>
                    </div>
                </div>

                <div className="navbar-collapse collapse w-100 order-3 dual-
collapse2 d-flex justify-content-end">
                    <ul className="navbar-nav ml-auto">
                        <span
                        className="nav-item nav-link text-info">
                            Fer
                        </span>
                        <button
                            onClick = {handleLogOut}
                            className="nav-item nav-link"
                            exact
```

```
                            to="/login"
                        >
                            Logout
                        </button>
                    </ul>
                </div>
            </nav>
        )
    }
//NOTA: en este script ya está hecho el ejercicio del segundo router
```

- Creo el DashboardRoutes.js en routers
- Hago las importaciones necesarias, necesito todas las rutas excepto el login
- Añado un HeroeScreen

```
import {Routes, Route} from 'react-router-dom'
import {MarvelScreen} from '../components/Marvel/MarvelScreen'
import {DcScreen} from '../components/DC/DcScreen'
import {Heroe} from '../components/Heroe/Heroe'
import {SearchScreen} from '../components/Search/SearchScreen'
import {Navbar} from '../components/UI/NavBar'
import{HeroeScreen} from './components/HeroeScreen.js'

export const DashboardRoutes = () => {
  return (
    <>
      <Navbar />

      <Routes>
            <Route path="/Marvel" element={<MarvelScreen />} />
            <Route path="/Dc" element={<DcScreen />} />
            <Route path="/Search" element={<SearchScreen />} />
            <Route path="/" element={<MarvelScreen />} />
            <Route path="/Heroe" element={<Heroe />}  />
        </Routes>
    </>
  )
}
```

- el AppRouter quedó así, sin el navbar ni las rutas
- Solo el login y creo una nueva ruta
- Estoy diciendo que todas las otras rutas desde / pasan por el DashboardRoutes
- El BrowserRouter no se coloca en rutas hijas

```
import React from 'react'
import {Routes, Route, BrowserRouter} from 'react-router-dom'
import {LoginScreen} from '../components/login/LoginScreen'

import { DashboardRoutes } from './DashboardRoutes'
```

```
export const AppRouter = () => {
  return (
      <BrowserRouter>

        <Routes>


            <Route path="/login" element={<LoginScreen />} />
            <Route path="/*" element={<DashboardRoutes />} />


        </Routes>


    </BrowserRouter>

  )
}
```

Navigate y useNavigate

- Ahora voy a gestionar la navegación del login al navbar e inversa
- Creo un botón para el Login
- Los hooks facilitan el poder extraer lógica de una manera muy sencilla
- useNavigate me va a regresar el navigate. Me permite navegar a otras pantallas
- En el handleLogin uso navigate y especifico la ruta
- replace en true para que reemplace el history y no poder volver atrás despues del login

```
import React from 'react'
import { useNavigate } from 'react-router-dom'

export const LoginScreen = () => {

  const navigate = useNavigate();

  const handleLogin =() =>{
    navigate("/Marvel", {
      replace: true  //que en lugar de hacer una nueva entrada ne el history la
reemplace
    })
  }

  return (
    <div className="container mt-5">
        <h1>Login</h1>
```

```
        <button className="btn btn-primary"
        onClick ={handleLogin}
        >Login</button>
      </div>

    )
}
```

- Ahora creo el handleLogOut, que ya se veia en eel navbar anteriormente y envío la ruta
- Ver NavBar anterior
- Lo que va a servir realmente para navegar de esta manera son rutas privadas y rutas públicas

CSSGRID

- display: grid;
- grid-template-columns: repeat(6,1fr);
- grid-template-rows: 1fr 1fr 1fr 1fr 1fr 1fr;

- grid-row-start: 2;
- grid-row-end: 3;

shorthand : - grid-row: 2 / 3

- Combinando rows y columns ubico el elemento dónde yo quiero en el grid

```
.box:nth-child(2){
    grid-row: 6/7;
    grid-column: 2/3
}
```

- Span agranda el elemento según le indiques. Ej:
    - Toma las 4 columnas y ocupa el espacio de estas con una caja flexible

```
.box:nth-child(1){
    grid-row: 1/span 2;
    grid-column: 3 /span 2
}

/*Es lo mismo que decir:*/

.box:nth-child(1){
    grid-row: 1/3;
    grid-column: 3/5
}
```

GRID SHORTHAND

- grid: 1-rows 2-columns
- Ex: 3 rows y 2 columns

```
.box{

    - grid: repeat(3,1fr) / repeat(2 1fr)

}
```

- Grid Autoflow:
    - Esta propiedad evita tener espacios libres cuando mueves cajas. Tiene varias opciones: dense, row dense, etc

```
.box{

    - grid: repeat(3,1fr) / repeat(2 1fr)
    - grid-auto-flow: dense
}
```

- Grid gap es para separar contenido, como un margin

```
.container{
    display: grid;
    column-gap: 1rem
}
```

- El shorthand es gap

GRID AREAS

- Se pueden nombrar las áreas
    - grid-template-areas
- De esta manera la caja 1 ocuparía toda la zona del header, y la 2 la del nav

```
.container{
    display: grid;
    height: 30rem;
    grid-template-areas: "header header header"
                         "nav nav nav"
                         "contenido contenido sidebar"
                         "footer footer footer";

    grid-template-columns: repeat(3, 1fr);
```

```
}

.box:nth-child(1){
    grid-area: header;
}

.box:nth-child(2){
    grid-area: nav
}
```

- Puedo añadir las rows para mejorar el diseño.
- Si lo añado en fracciones no tengo que preocuparme de los margin añadidos con gap ni el total del espacio repartido

```
.container{
    display: grid;
    height: 90rem;
    grid-template-areas: "header header header"
                         "nav nav nav"
                         "contenido contenido sidebar"
                         "footer footer footer";

    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: 2.5fr 1fr 6fr 2.5fr
}

.box:nth-child(1){
    grid-area: header;
}

.box:nth-child(2){
    grid-area: nav
}
```

Grid Template

```
 Grid Template incluye las areas las rows y las columns
```

```
.container{
    grid-template: "header header header" 2.5fr
                   "nav nav nav" 1fr
                   "contenido contenido sidebar" 6fr
                   "footer footer footer"2.5fr / 1fr 1fr 1fr
```

```
}
```

- El repeat no funciona aqui

- Alineación en Grid es muy similar a flexbox
- además de align items, tengo place-content y align-content

AutoFIll y AutoFit

-Auto-fIll continua agregando espacio si lo hay, sin las cajas -Auto-fit va a generar una nueva columna para un añadido, y maneja solo el espacio dispuesto sin especificarlo

```
.container{
    display: grid;
    grid-template-columns: repeat(auto-fill, 200px)
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr))
    grid-template-columns: repeat(auto-fit, 300px, 1fr)
}

----
```

Selector de dos clases

clase1.clase2

Selector de 2 clases anidadas

.clase1 .clase2

Etiqueta y clase

div.clase

Seleccionar 2 elementos

.texto, .otro_elemento

Seleccionar 1er nivel de hijos

.padre > hijo

Seleccionar 1er elemento despues del selector

.padre tag_hijo + el_siguiente_elemento

Seleccionar por tributo

- Con $= tiene que terminar con...
- Que empiecen por.. https a[href$=".com"] a[href^="https"] input[type="text"]

El primer hijo de una lista

ul li:first-child ul li:first-of-type

El ultimo elemento de una lista

ul li:last-child

Un elemento en especifico

ul li:nth-child(2)

- Selecciona el segundo elemento

ul li:nth-child(2n+1)

*20 +1= 1 2*1 +1= 3 2*2 +1= 5

Así obtengo los impares

Todos menos.. (la excepcion)

p:not(.texto)

p:not(.texto):not(.oferta)

Primera linea de un párrafo

.clase::first-letter{ font-size: 20rem } .primera-linea::first-line{ font-size: 20rem }