FRONT END III

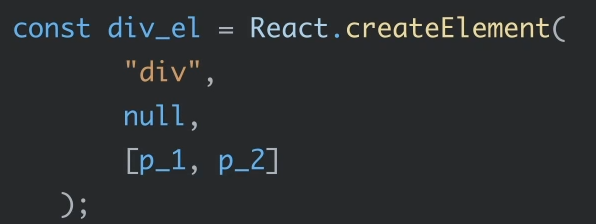
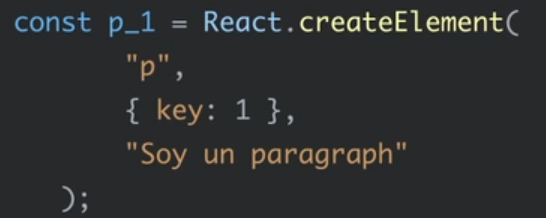
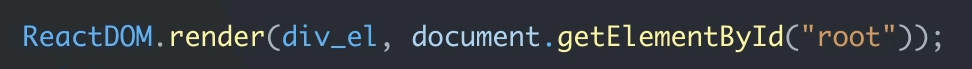
*Micaela Feller (tutor), Julian Debrabandere (docente), Ivan Szucs (ayudante técnico)*

CLASE 1

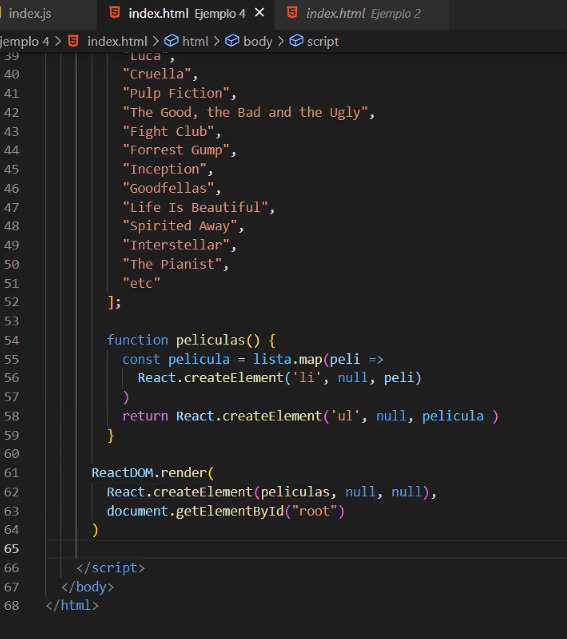
Frameworks: Vue.js, AngularJs

Librería: React

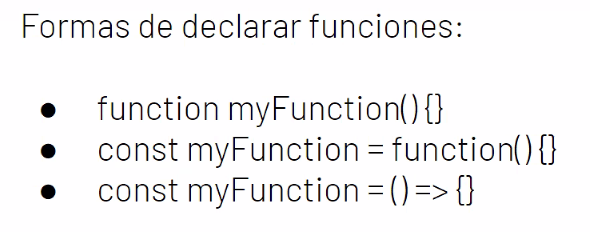
Formas de crear elementos:

* React.createElement (type, [use], […children]);
  + Por ejemplo: 
  + 
  + Y luego se renderiza con el dom 
  + Necesita que se agreguen los scripts para que renderice: *<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>*

*<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>*

* + Ejemplo final 
  + Otro ejemplo: dentro del script creo la const con el listado, luego recorro ese listado para que se cree el <li> y luego lo renderizo
  + 
* JSX = JavaScript XML

CLASE 2



Para trabajar con JSX:

* Agregar *<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>*

*<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>*

*<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>*

* Y todo el código se escribo dentro de *<script type="text/babel">*

Pero de esa forma, es más lenta la web. Entonces, también puedo:

* Ejecuta npm install babel-cli@6 babel-preset-react-app@3
* npx babel --watch src --out-dir . --presets react-app/prod

Para declarar objetos en JS:

Const objeto1 = {

Atributo: valor,

Atributo: ‘valor’ }

Si quiero recuperar valores de un objeto y agregarlo algo más, uso spread operator:

Const nuevoObjeto = {

…objeto1,

Atributo: valor,

Atributo: ‘valor’ }

CLASE 3

Explicación de código:

* creo el array a iterar
* creo la constante que creará lo que necesito en HTML
* guardo en una variable la iteracción con map
* devuelvo la sección con la variable dentro
* luego renderizo la constante creada para el HTML. Recordar que es un componente y necesita que los signos se cierren en el mismo nombre del componente.



CLASE 4

CRA = Create React App = configura ambiente de desarrollo con servidor y soporte de aplicación web, con linting (detección de errores de sintaxis, código difícil de mantener, malas prácticas).

Para instalarlo:

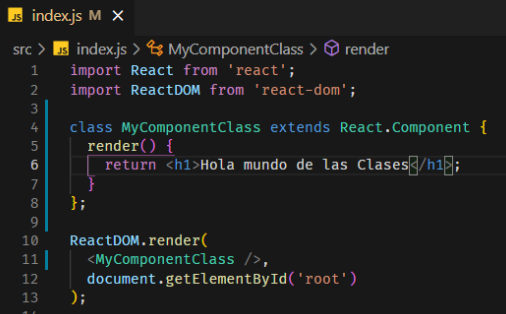
1. npm init react-app “proyecto”
2. cd “proyecto”
3. npm start
4. queda en localhost:3000

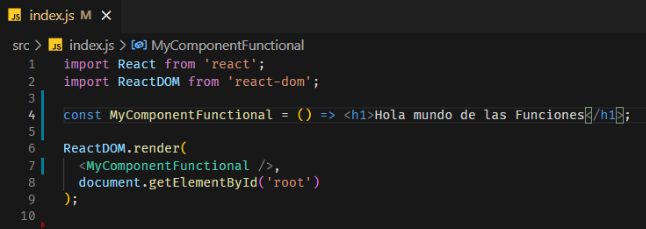
O puedo simplemente:

1. npx create-react-app [nombre del proyecto]
2. cd [proyecto]
3. npm start

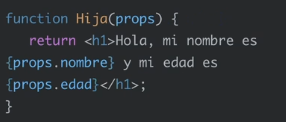
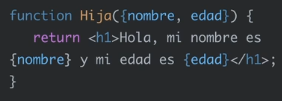
CLASE 5

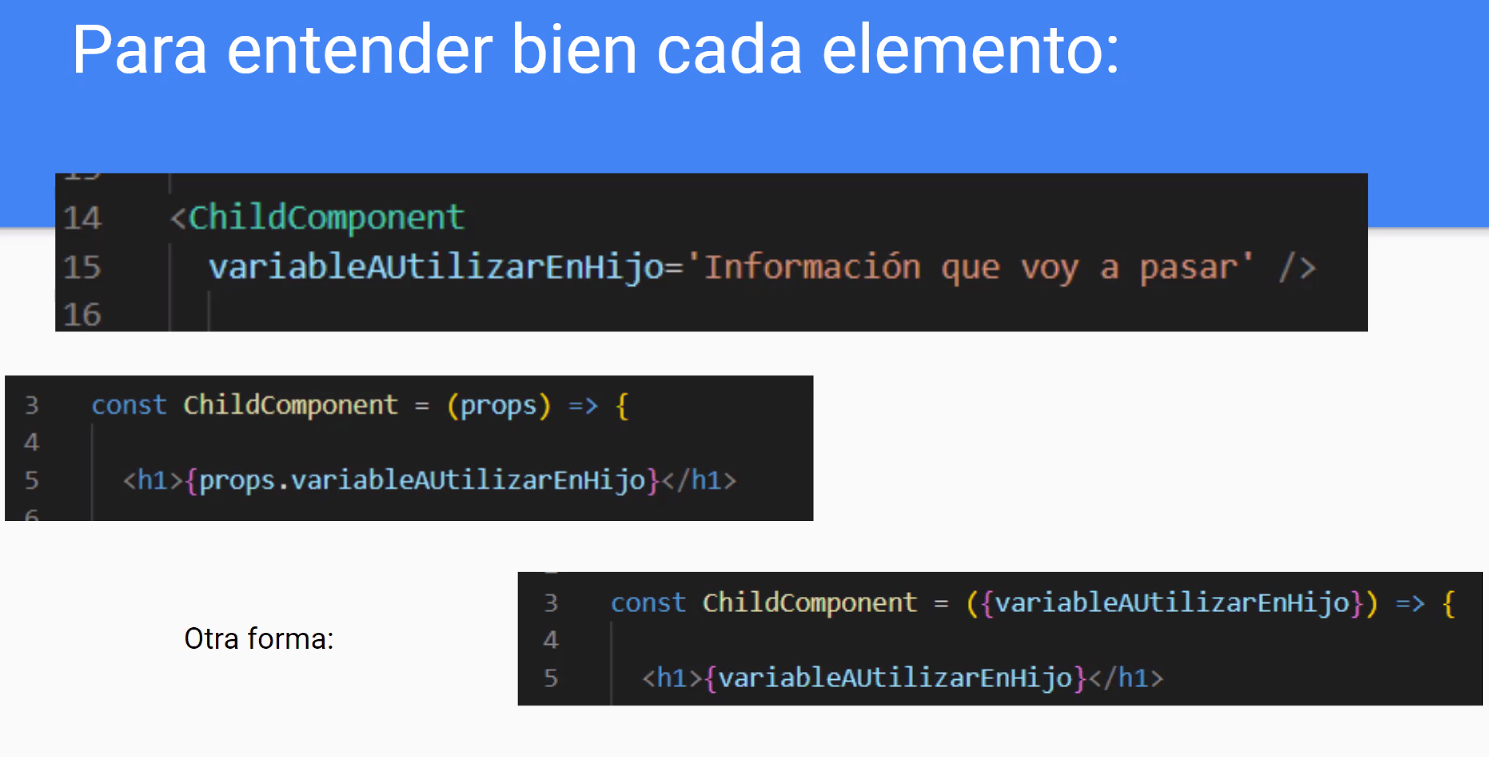
Componente = clase o función que devuelve HTML a través de sintaxis JSX. Existen componentes de clase y componentes funcionales.



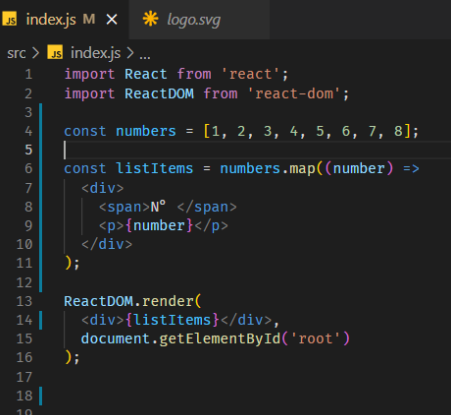
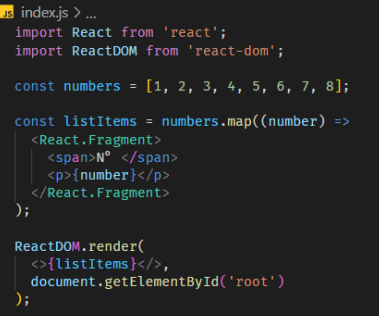


Las Props permite desestructurar los componentes (pasar información de un componente a otro):

o 

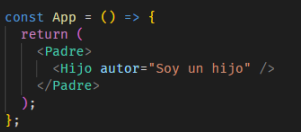


Fragment, permite que cuando se renderice el código, no se cree cantidad de divs vacíos. Y simplemente se usa poniendo <React.Fragment></React.Fragment> como si se tratara de un div.

 🡪 

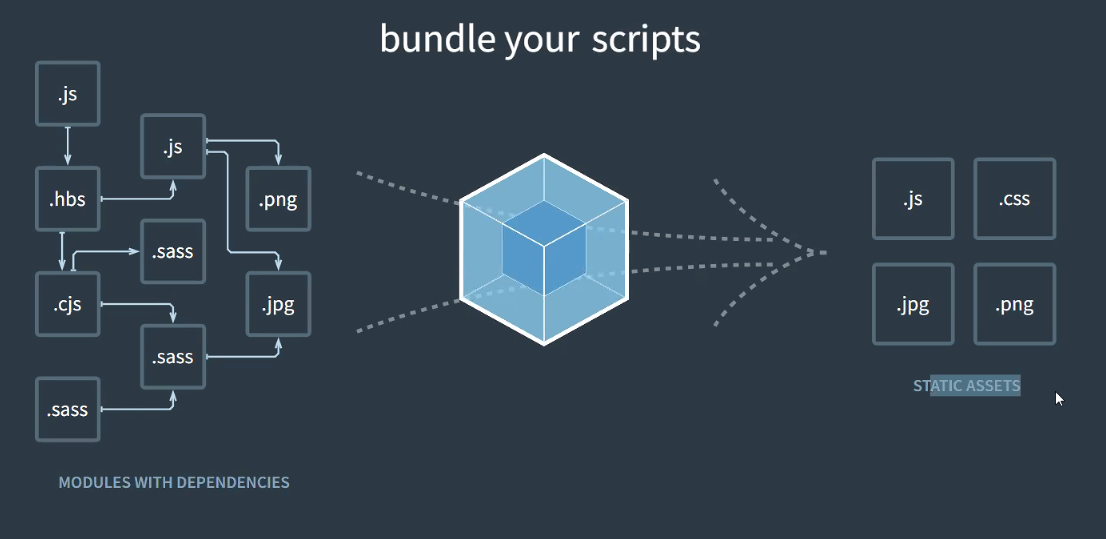
Children: todo lo que va en los componenten debe estar definido desde el padre. Pero si no sé que hijo va a tener, puedo poner simplemente {props.children} y con eso me estoy asegurando que puede ser de cualquier tipo el hijo.





CLASE 6

Webpack ordena todo para que el browser lo interprete como un solo documento y no como dependencias.



Simple React Snippets

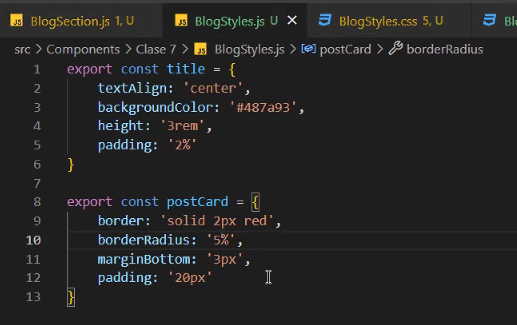
* rafce 🡪 crear un componente arrow function exportado
* rcc 🡪 crea un componente de clase exportado

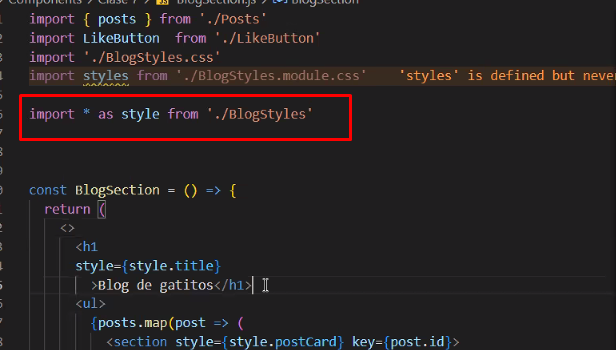
CLASE 7

Map: crea un nuevo array

Key: es de tipo string

Módulos CSS: consumidos como objetos JS, automatizan nomenclaturas de clases y animaciones. Es decir, importo con el nombre estilo (o cualquier) todo el archivo scss o css. Y luego, para cada elemento le tomo el estilo que quiero. Es importante que en el style, exporte cada estilo:







Componentes estilizados: explotan las tagged templates y eliminan el mapeo entre componentes y estilos.

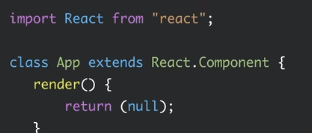
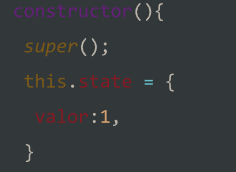
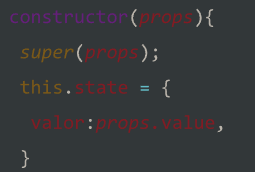
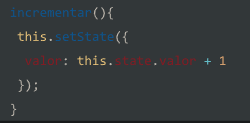
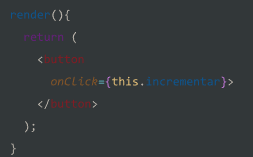
Para trabajar con estilos in-line, va escrito con CamelCase y el valor entre comillas:



CLASE 8

Estados: programas que manejan sus propios datos y los protegen para que no sean modificados. A diferencia de los props (que pueden cambiar desde afuera lo que hace un componente), el estado solo lo puede cambiar el mismo componente.

Para definir una clase:

* class nombre
* extends React.Component
* solo puede tener el método render ()
* que devuelve un return
* 
* Luego se pueden desarrollar funciones que definen los estados.
* Creo el constructor con super() y le asigno los estados (clave/valor)
* 
* Pero tambien podemos combinar con el uso de props y asi obtener datos del padre
* 
* Y luego, para modificar los estados, siempre se usa this.setState({ lo que debe hacer})
* 
* Y luego defino ante qué evento va a cambiar
* 

IMPORTANTE: para importar datos desde un json:

Import \* as dato from ‘../data/datos.json’;

Otra forma es que dentro del constructor ya le aclare cuales serán sus funciones que pueden ser llamadas por otros componentes agregando bind:

This.nombreDeFuncion = this.nombreDeFuncion.bind(this)

Y debajo escribo la funcion nombreDeFuncion y todo lo que hace.



Formas de exportar:

* Export { Pelicula }
  + Lo que hace es exportar a la clase pelicular, porque tal vez el archivo tiene varias clases.
  + El nombre debe ser exacto al de la clase
  + Y se importa exactamente igual
    - Import { Pelicula } from ‘…..’
* Export default Pelicula
  + Exporta todo
  + El nombre no es relevante
  + Se puede importar con cualquier nombre
    - Import Movie from ‘…’

GIT HUB

1. Clonar repo: git clone https://github.com/nico-limo/frontend\_3-clase\_9-repaso.git
2. Entrar a la carpeta: cd frontend\_3-clase\_9-repaso
3. Creo lo falta en la terminal de visual studio code: npm i
4. Npm start

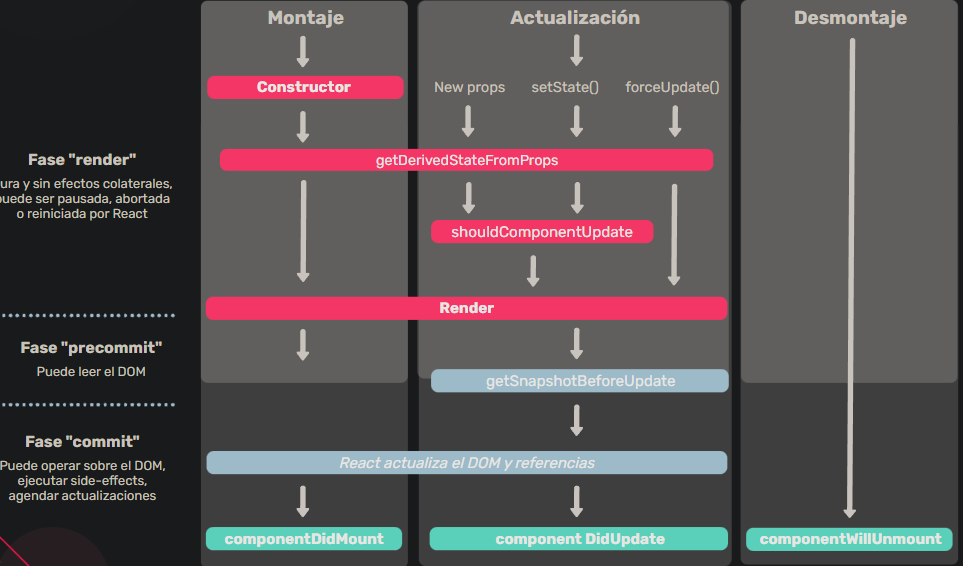
SI QUIERO SUBIR A GIT HUB

1. Creo el repositorio web
2. En la terminal de Visual Studio git init
3. Git add .
4. git commit -m "first commit"
5. git branch -M main
6. git remote add origin <https://github.com/AnaLauraS/Examen1-FrontEndIII.git>
   1. si falla el add origin uso:
   2. git remote set-url origin git@github.com:ppreyer/first\_app.git
7. git push -u origin main

CLASE 10

Ciclo de vida de un componente:

* Montaje 🡪 constructor y componentDidMount
* Actualización 🡪 render (con New props, setState y forceUpdate) y componentDidUpdate
* Desmontaje 🡪 componentWillUnmount



CLASE 11

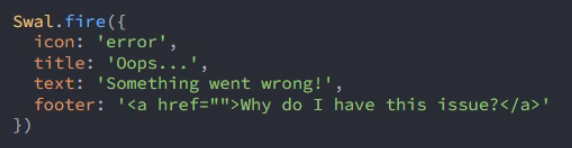
Eventos: onClick, onSubmit, onChange.

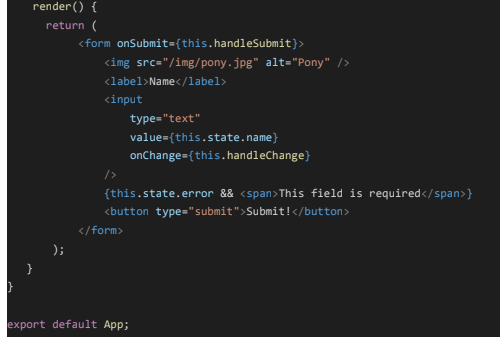
* Componentes no controlados: controlados por DOM
* Componentes controlados: controlados por React (por state, props y eventos)

Formik: librería de formularios

sweetAlert: npm install –save sweetalert2, o sin el --save, y luego importarlo



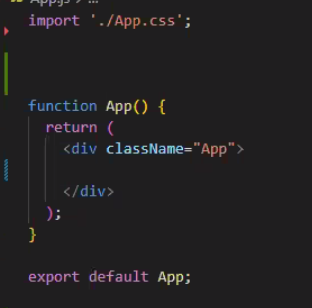


**TIPS**: Para seleccionar toda las palabras y modificarlas al mismo tiempo:

* Selecciono una
* Ctrl + D
* Eso selecciona todas
* Modifico lo que me interesa

**TIPS**: no usar más clases, sino funciones:



**TIPS:** Para clases, si no paso props y uso solo arrow functions, no necesito constructor ni bind.

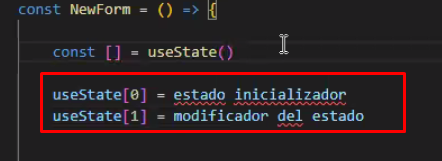
**TIPS**: condición ternaria: solo si se da true el primero, hace el segundo



**TIPS**: crear componente de funcion con RAFCE directamente.



Armar **funciones** en vez de clases, con Hooks:

 🡪



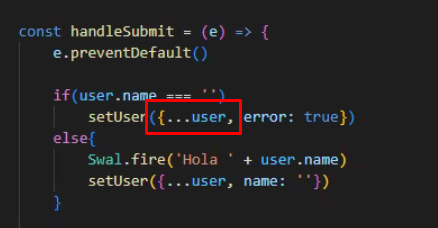
El primero es la variable y el segundo la función que lo modifica.

Entre paréntesis pongo el tipo de la variable (si es objeto va {}, si es array []): 

Para anotar las funciones que van a usar las funciones, uso const, y llamo a la funcion en el elemento que reacciona dentro del return:



Cuando trabajamos el estado como objeto, cada vez que seteo una propiedad, me elimina todas las otras que no modifico. Entonces para evitar eso, uso spread operator de puntos, y el valor que actualizo:



Examen

<https://github.com/Frontend-III/entregable-frontend-3-junio22>

repo con clases <https://github.com/juliandevbra/proyecto-camada-8>

CLASE 14

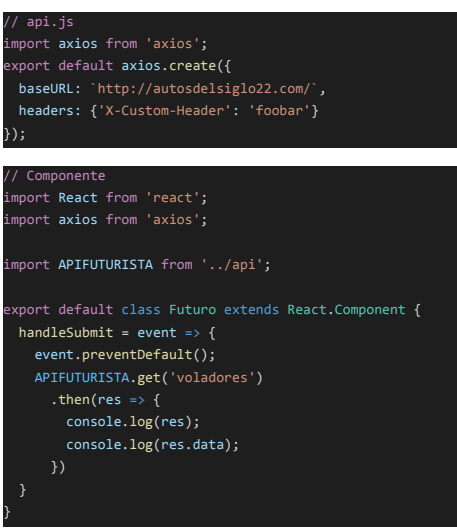
Efectos secundarios: operación invocada dentro un ámbito específico pero que transcurre fuera del mismo 🡪 el resultado es difícil de manejar (como las APIs).

Principio de Responsabilidad Única (SRP): una clase o módulo debe tener solo una razón para cambiar.

Axios es una biblioteca que se requiere instalar, para trabajar con APIs y sus promesas. Convierte los JSON automáticamente. Se basa en Node.js 🡪 npm install axios



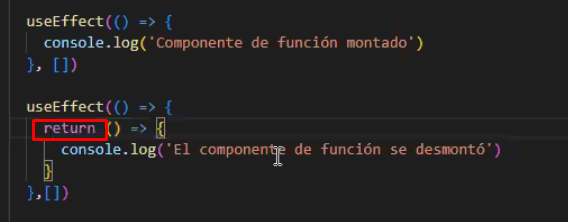




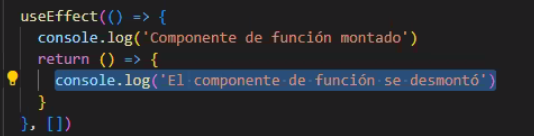
Para informar lo de montar y desmontar componentes (eso solo funciona en componente de clases) en componentes de tipo de función: USEEFFECT

* useEffect(()=>{ funcion }, []}
* primero tiene una arrow function
* y luego, entre corchetes, las dependencias.

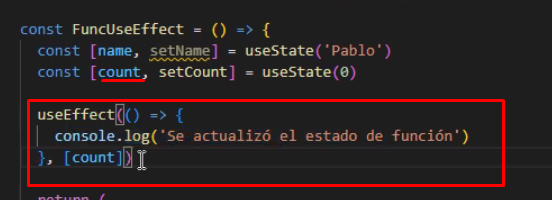
El useEffect se usa para cuando se monta y cuando se desmonta. Para diferenciarlos, en el desmonte, se pone un return de otra arrow function.



Pero tambien se puede hacer dentro de un mismo useEffect

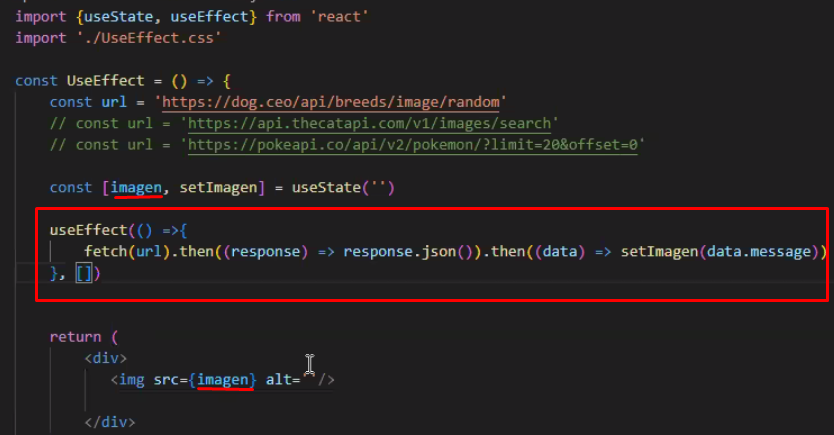


También puedo decirle que cada vez que se actualiza la “dependencia” (que es un estado), haga algo:

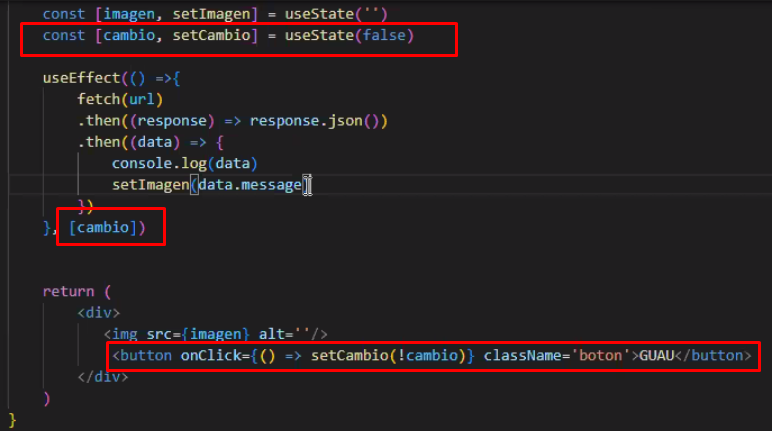


Para usarlo con APIs:





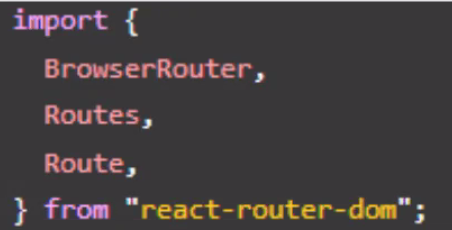
Puedo agregar como dependencia un booleano para que se active el useEffect:



CLASE 16

Instalar React Router v6:

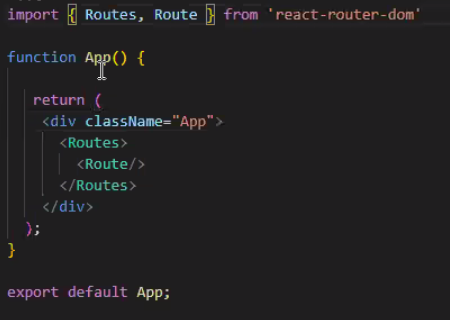
1. npm install react-router-dom@6
2. importar los 3 elementos:



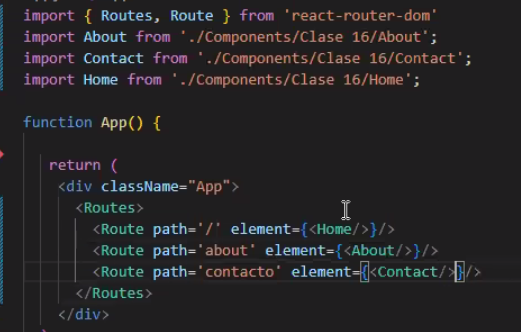
1. (se recomienda tenerlo en el index)



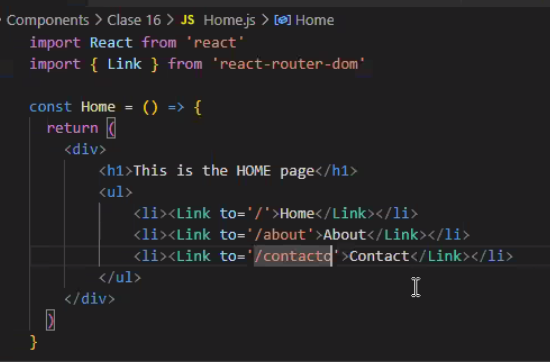
1. En el App pongo las rutas:



1. en la app principal,
   1. decimos cual es la pagina principal
   2. y las paginas secundarias que va a tener



1. Luego agrego los li de navegación en las páginas.



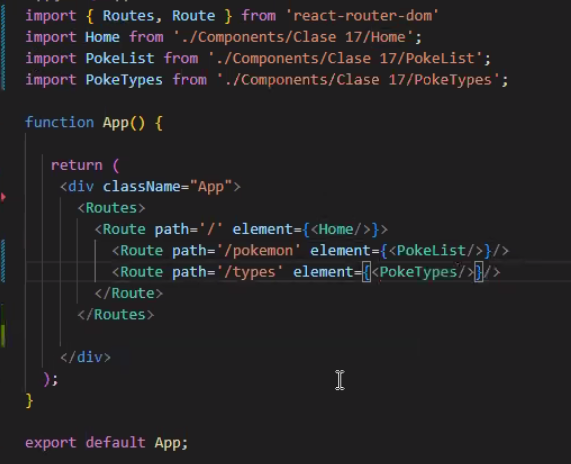
1. También lo puedo hacer con un onClick



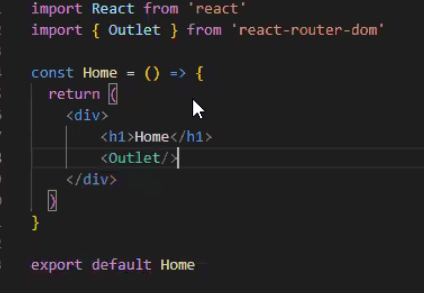
1. luego creo los otros componentes que serán las páginas normales.

CLASE 17

Anidar rutas: solo renderiza la principal.

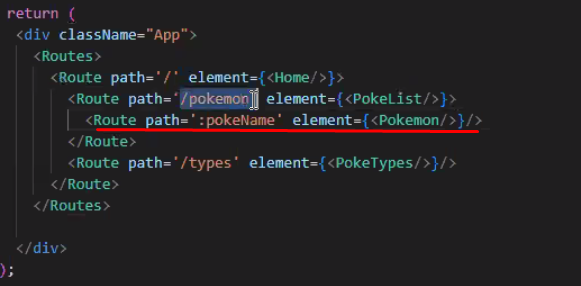


Pero para que funcione el ruteo, en la ruta principal debo poner “OUTLET”

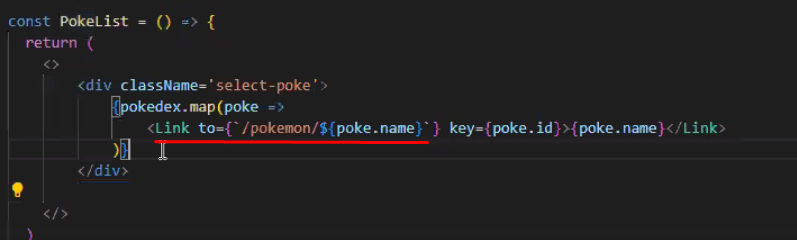


Para crear rutas por parámetros:

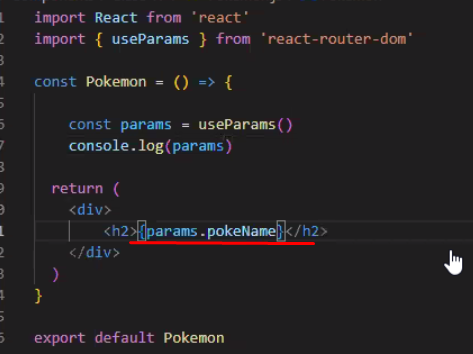
1. el route va a tener “:” para identificar que es un parámetro



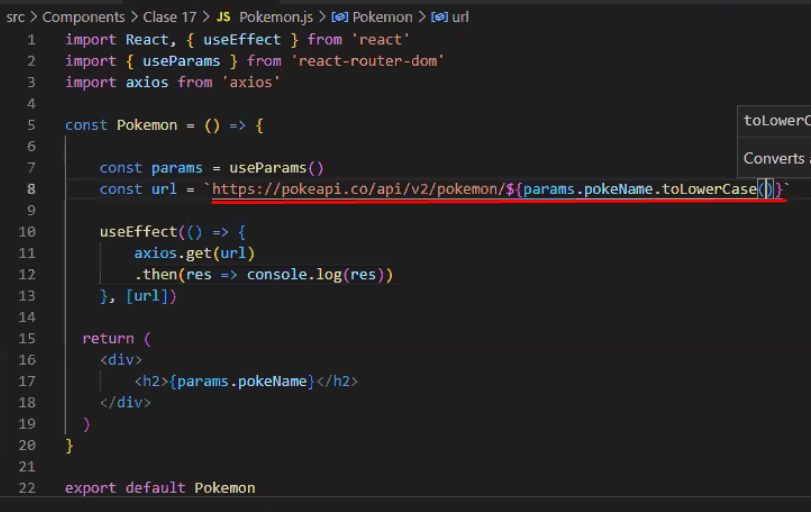
1. el Link to tiene que tener {} para informar que es JS, y dentro ´´ para informar que va a ir un template literal y el signo $ para informar que es una variable.



1. para que lo que le pase en route lo reciba y renderice, uso USEPARAMS (si lo quiero usar adentro, le tengo que poner . + el nombre de la variable definida en el route)



1. lo mismo que el 3, usando axios (no hace falta recuperar el json primero como con fetch):



Me conviene traer lo de la API en un estado:



CLASE 19

Hook = función especial que permite ‘conectarse’ a características de React, dotan a los componentes funcionales de estado interno y ciclo de vida. Permite insertar las siguientes características:

* estado
* ciclo de vida
* contexto
* referencias

Surgió Hook para solucionar:

* la lógica de estado no es fácil de reutilizar, porque no es fácil compartir estados entre componentes.
* los componentes complejos son difíciles de comprender
* difícil el testing
* la implementación de clases puede ser confusa y molesta

Entonces los hooks permiten:

* mejor y simple reutilización, composición y testeo
* extraer lógica de un componente para reutilizarla y compartirla

Beneficios Hooks:

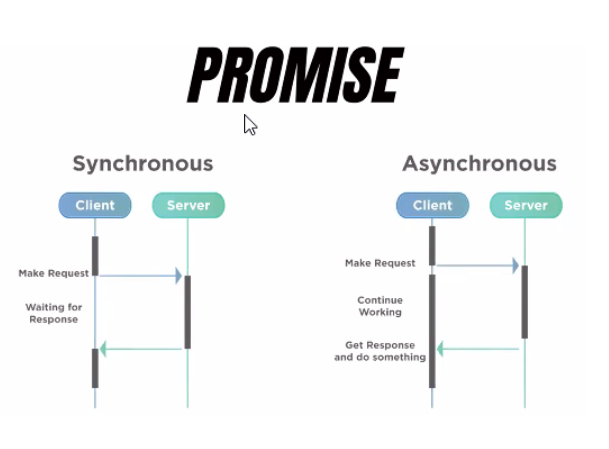
* menor cantidad de código
* código más organizado
* funciones reutilizables
* fácil testeo
* no llama a super()
* no usa this y bind
* componentes reducidos.

Reglas de hooks:

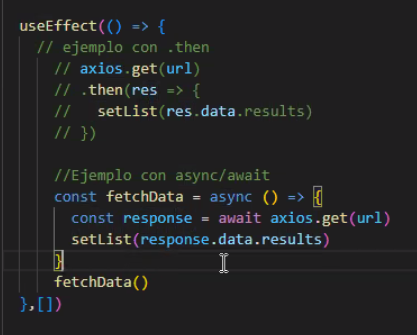
* no llamarlos dentro de condicionales, ciclos o funciones anidadas
* llamarlos solo dentro de componentes funcionales
* llamarlos al inicio de la función del componente

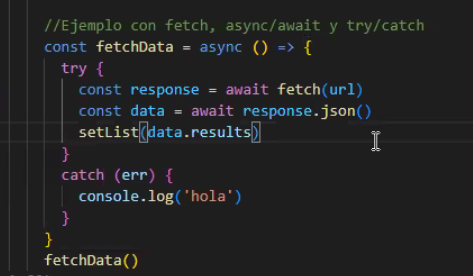
Hooks nativos:

* useDebugValue
* useState: Si alguna pieza de ese estado actualiza su valor, React volverá a renderizar los componentes involucrados con dicha pieza de estado tras hacer una comparación entre el virtual DOM y el DOM guardado en memoria. El objetivo de esto es que la actualización sea eficiente y basada en la diferencia entre ambos DOM. Para definir su estado necesitamos de una variable, una función y useState.
* useEffect: Para manejar el ciclo de vida. Hace que los efectos sean predecibles y controlables.
* useContext
* useReducer
* useImperativeHandle
* useMemo
* useRef
* useCallback
* useLayoutEffect









CLASE 23

La diferencia entre un Hook personalizado y un componente, es que el Hook no devuelve un html, sino un objeto.

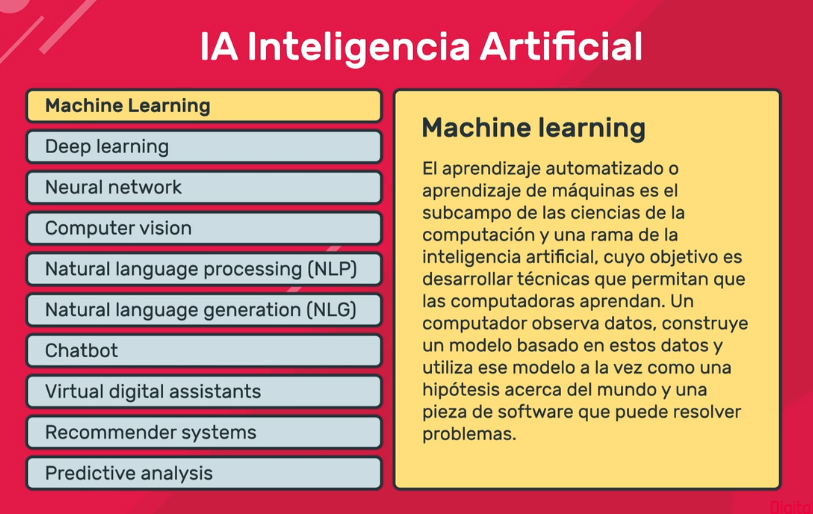
Luego creo las const que usarán ese hook personalizado, tantas veces como quiera.

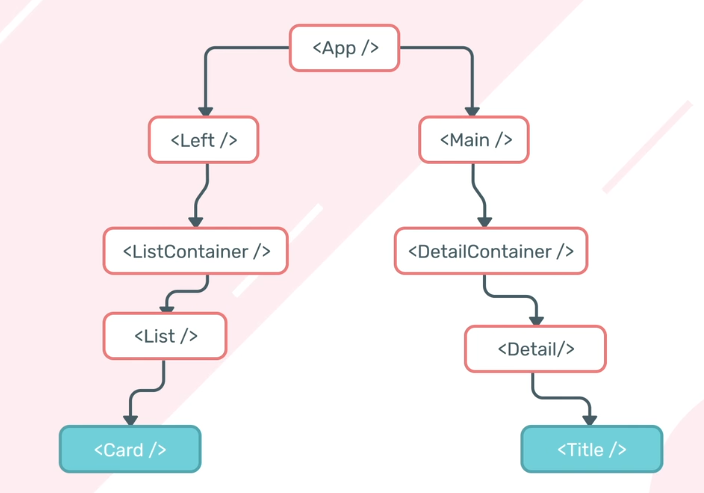


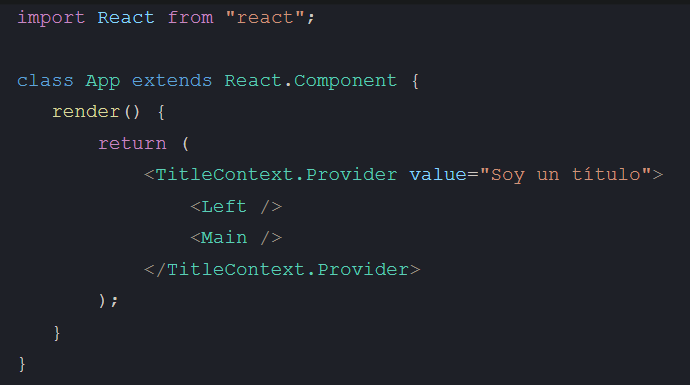
CLASE 25

Estados locales se comparten por props.

Estados globales: para compartir datos de un elemento a otro, siendo hermanos (o primos) 🡪 API context (es similar a Redux pero ésta se usa solo en proyectos super grandes y complejos)



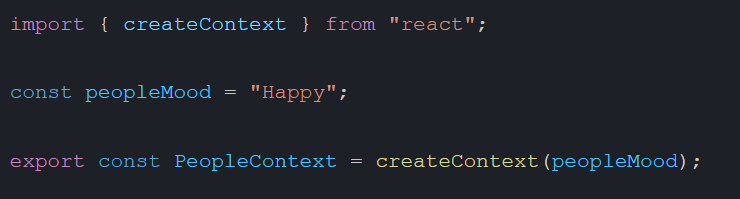
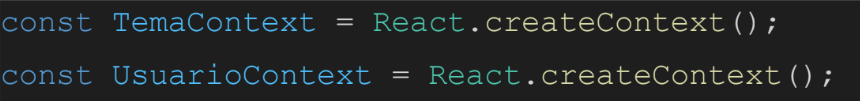
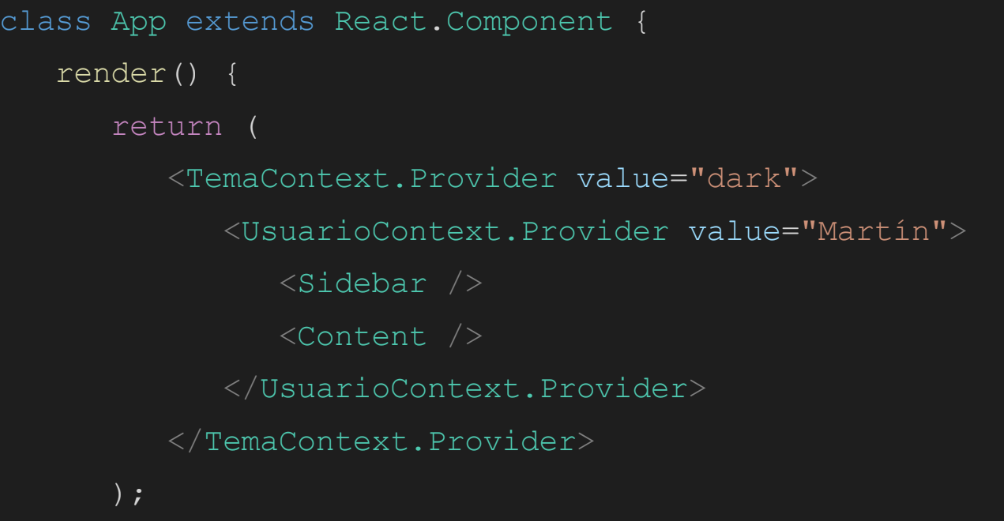
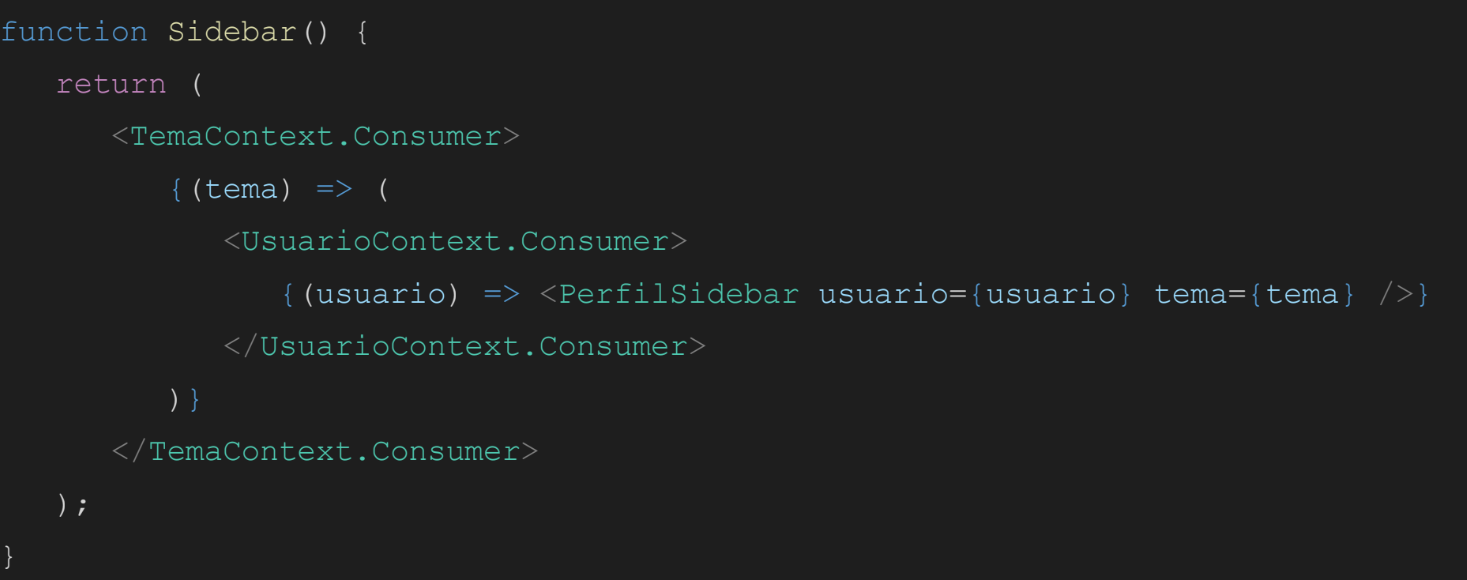




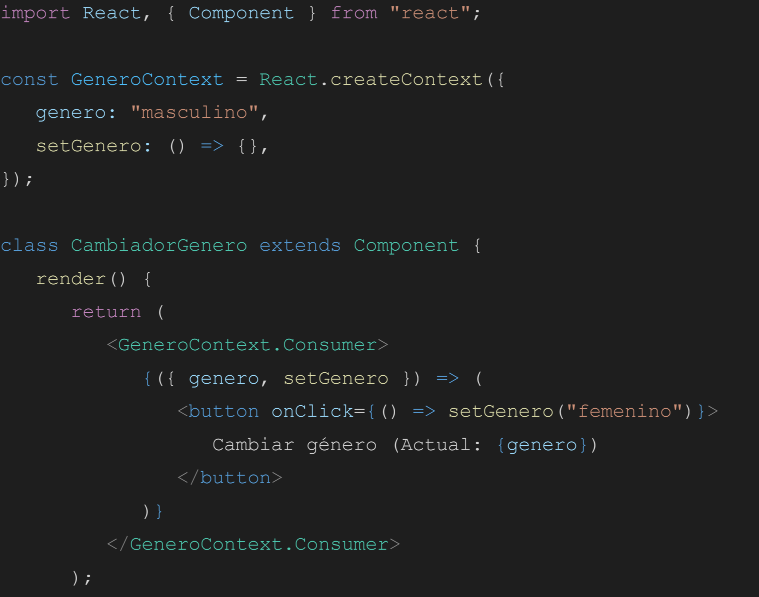
Se usa para:

* tema oscuro / claro
* autenticación de usuarios
* internacionalización (idioma)

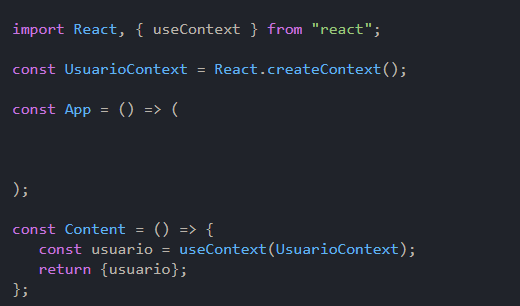
Pasos para implementar context:

* createContext
  + 
  + No es obligatorio pasarle un valor inicial.
  + 
* context Provider para encapsular un árbol de componentes
* usar value en el Provider
  + 
* leer la propiedad usando el context Consumer.
  + 

El context se cambia igual que los estados (solo para componente de clase):



Podemos usar useContext para devolver el valor actual.



**Resumen para componentes funcionales:**







CLASE 26

Storybook

React Bootstrap

Prop-types

Styled Components

Tailwind

Material UI (recomendado)

Ant (diseño de componentes)

