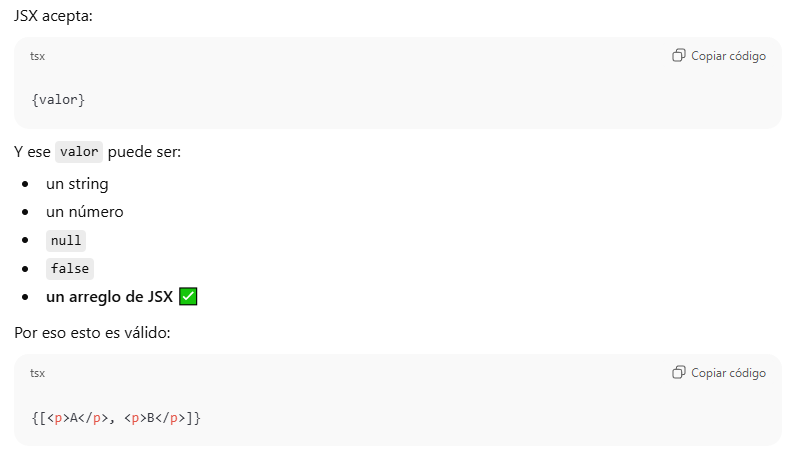
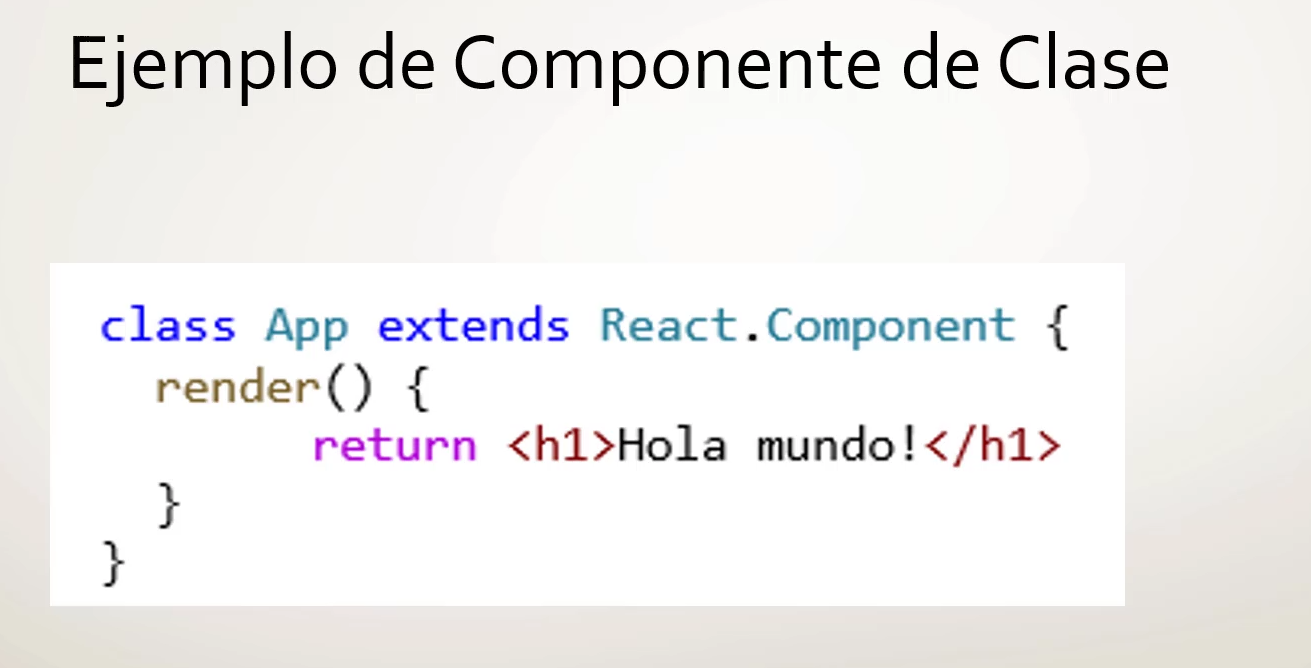
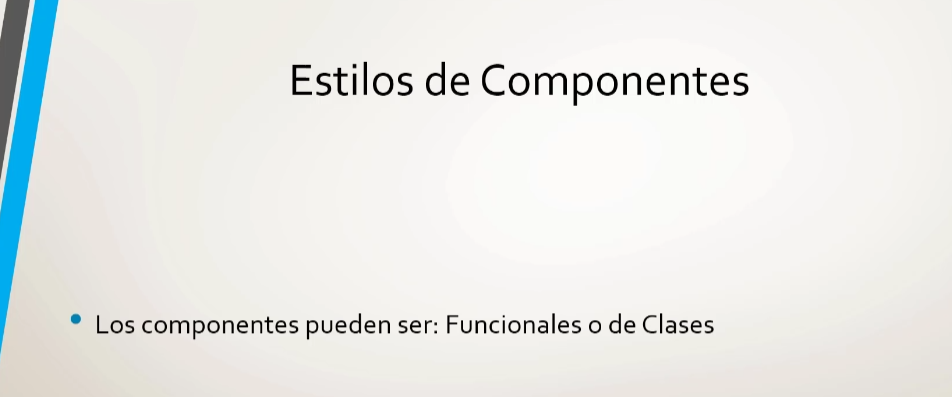
En el *return* se coloca código JSX, con sintaxis similar a HTML, y entre {} indicamos cuando quiero colocar código JS/TS, si se coloca un valor correspondiente a una variable declarada anteriormente, este tiene la siguiente característica:





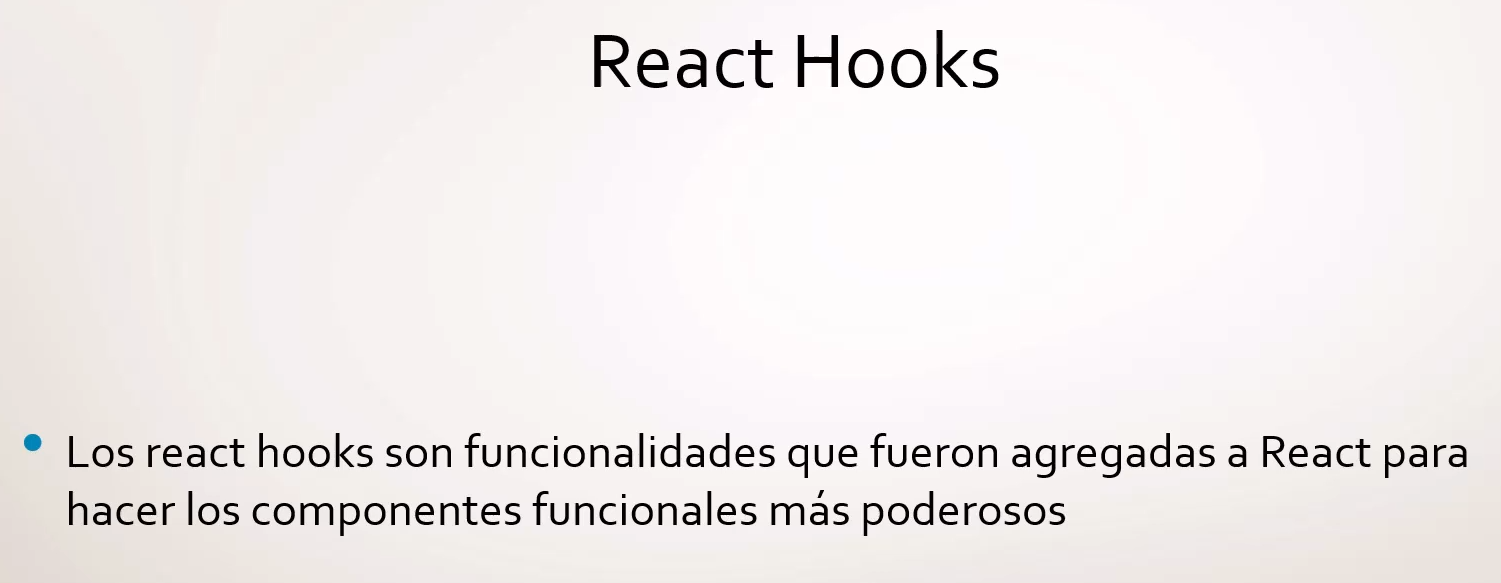


**React.Fragment — resumen breve**

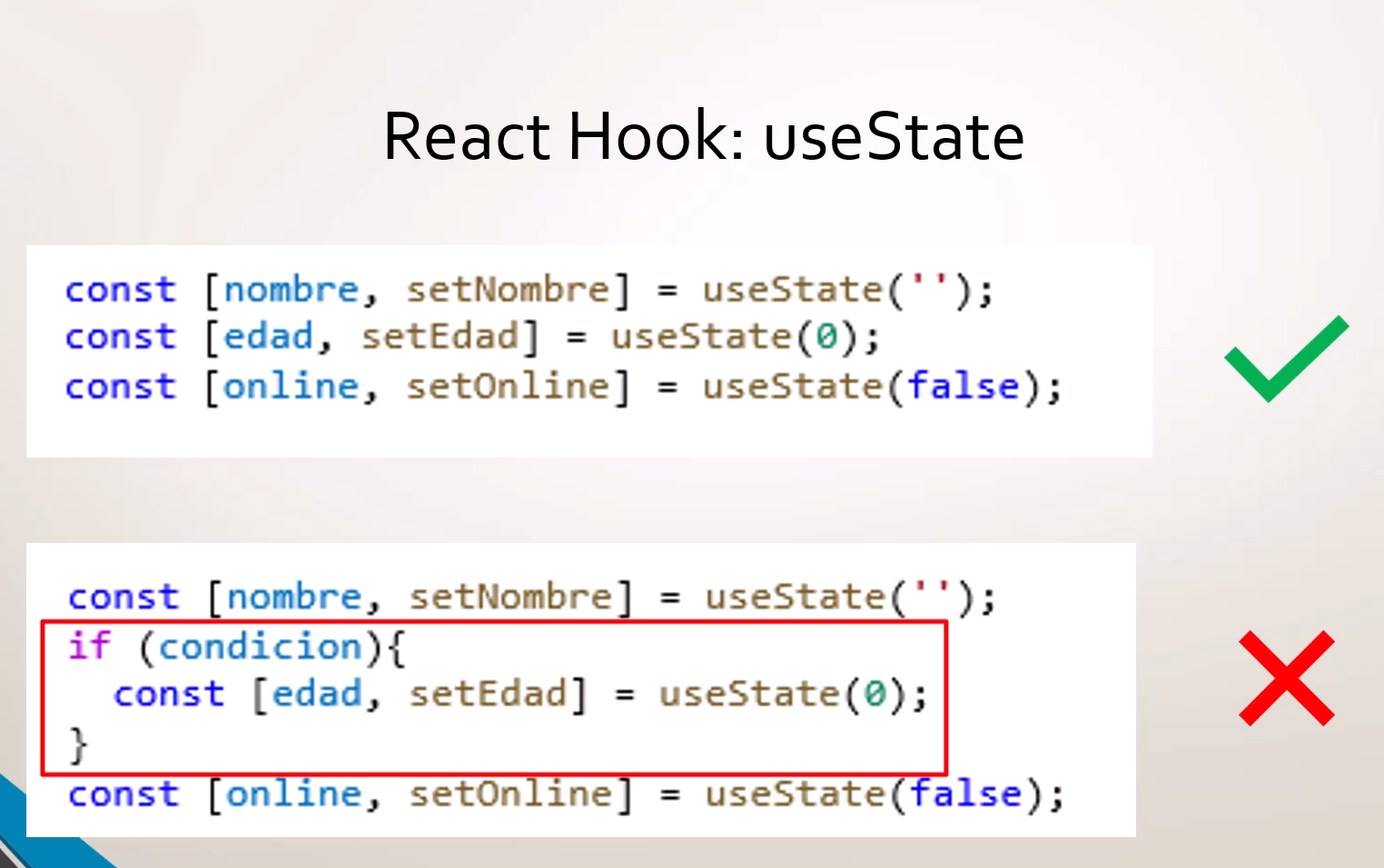
En React, **un componente debe devolver un solo padre**.  
Si tenés varios elementos hijos (h1, h2, etc.), necesitás un **padre común**.

**React.Fragment es un padre “invisible”**: cumple la regla de React **sin agregar un <div> al HTML**.



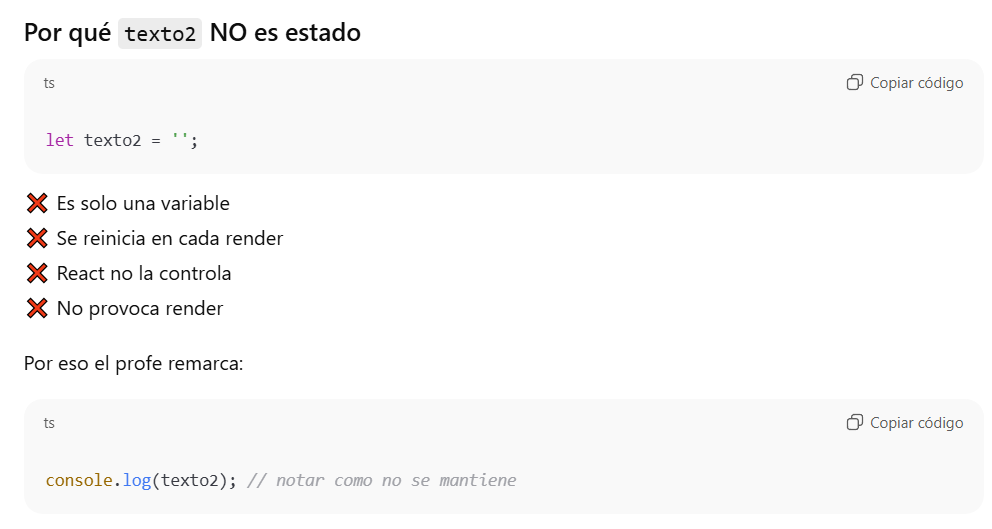
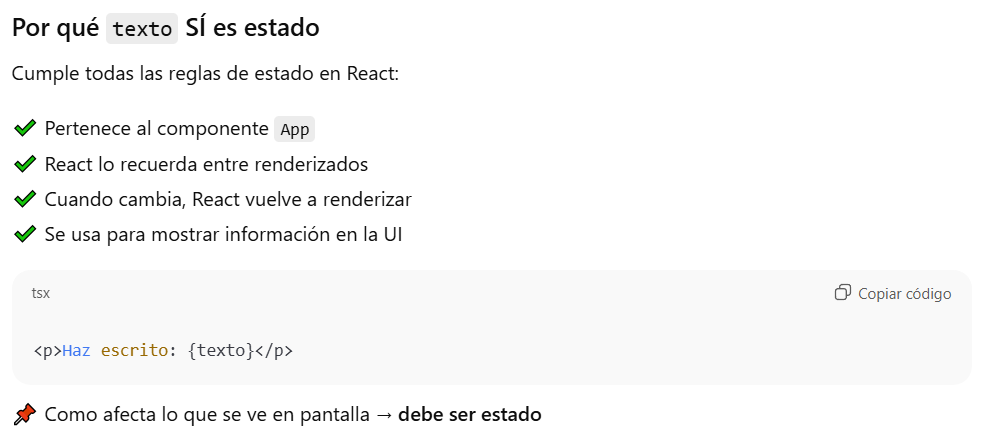


* De esta manera logramos rendedrizar el componente al modificar el estado y hacerlo ver reflejado.



Ejemplo:





**Por lo tanto, useState es un hook que permite a los componentes funcionales definir estado interno, el cual es persistido por React entre renderizados y cuya modificación provoca la re-renderización del componente**.

**Interfaces + Pasar parámetros a los componentes**

export default function MostrarTexto(props: MostrarTextoProps) {

    return (

        <div>

            <p>Haz escrito: {props.texto}</p>

        </div>

    );

}

interface MostrarTextoProps {

    texto: string;

}

/\*

--- Esta es laforma de trabajar con interfaces en React + Typescript ---

Typescript hace esto con las interface automáticamente:

props = {

  texto: "algo que viene del componente padre"

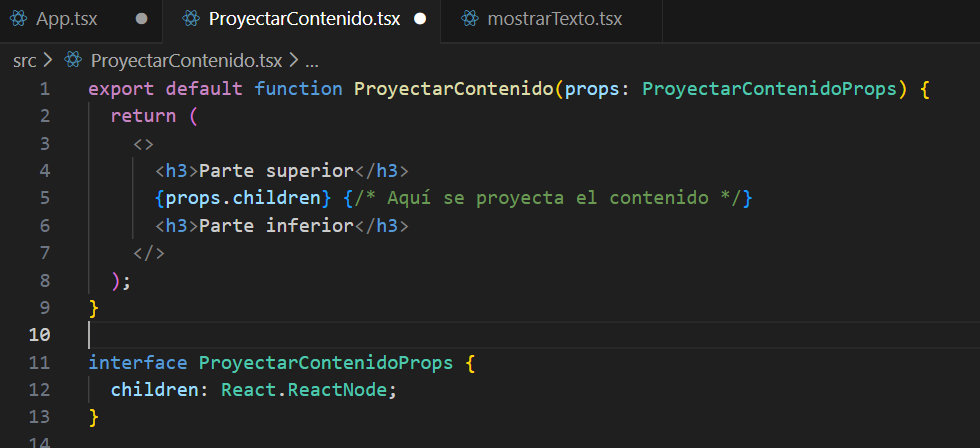
}

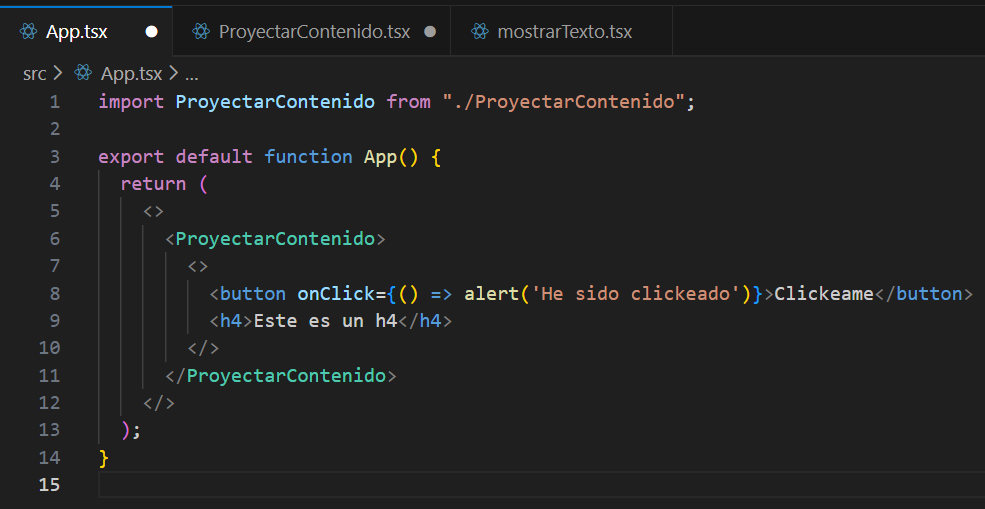
Por eso despues usas:

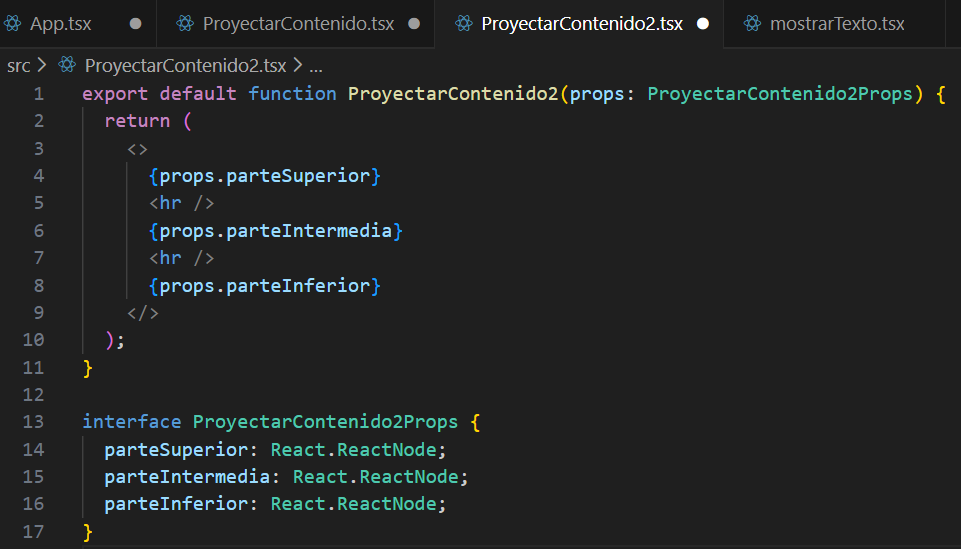
props.texto

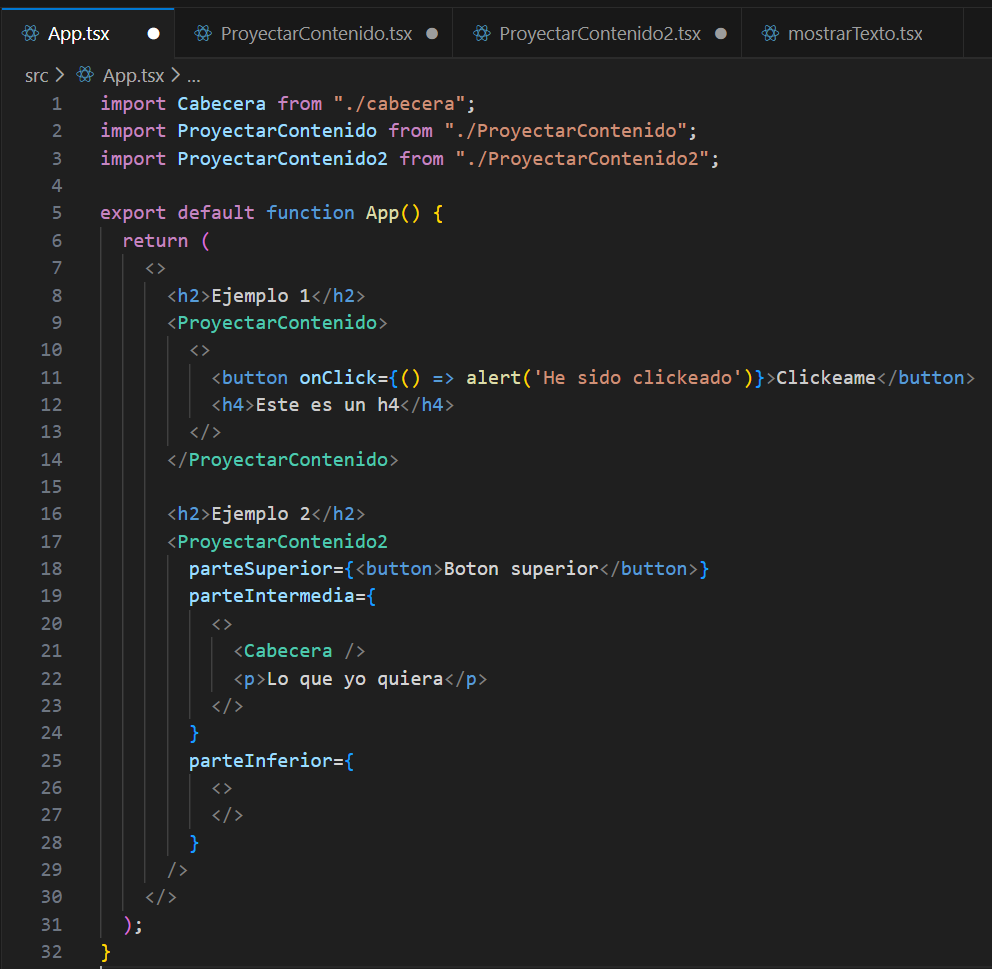
\*/

**Se puede proyectar el contenido de un componente en otro: de 2 formas:**

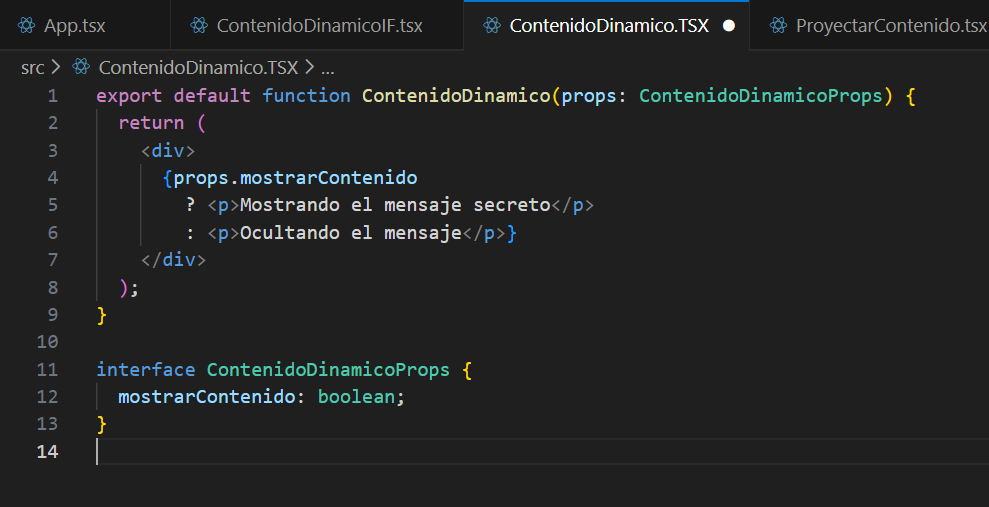
**Con Children**

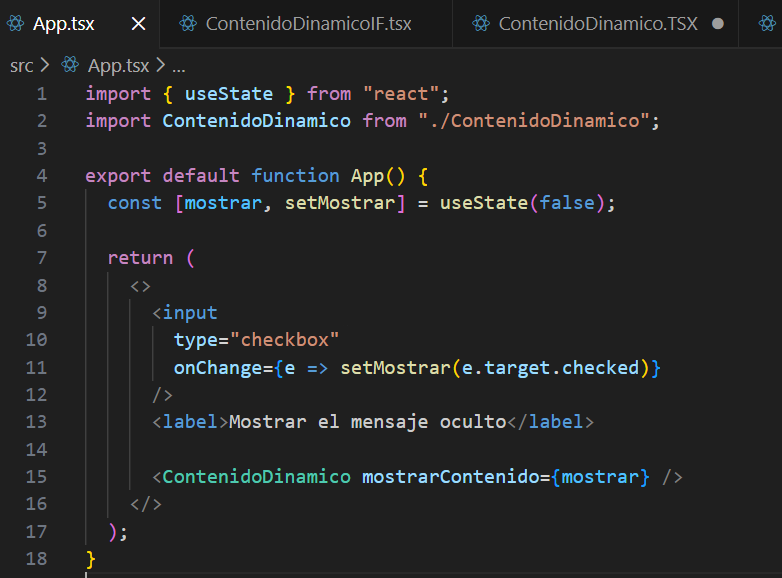


**Con atributos en el componente:** 

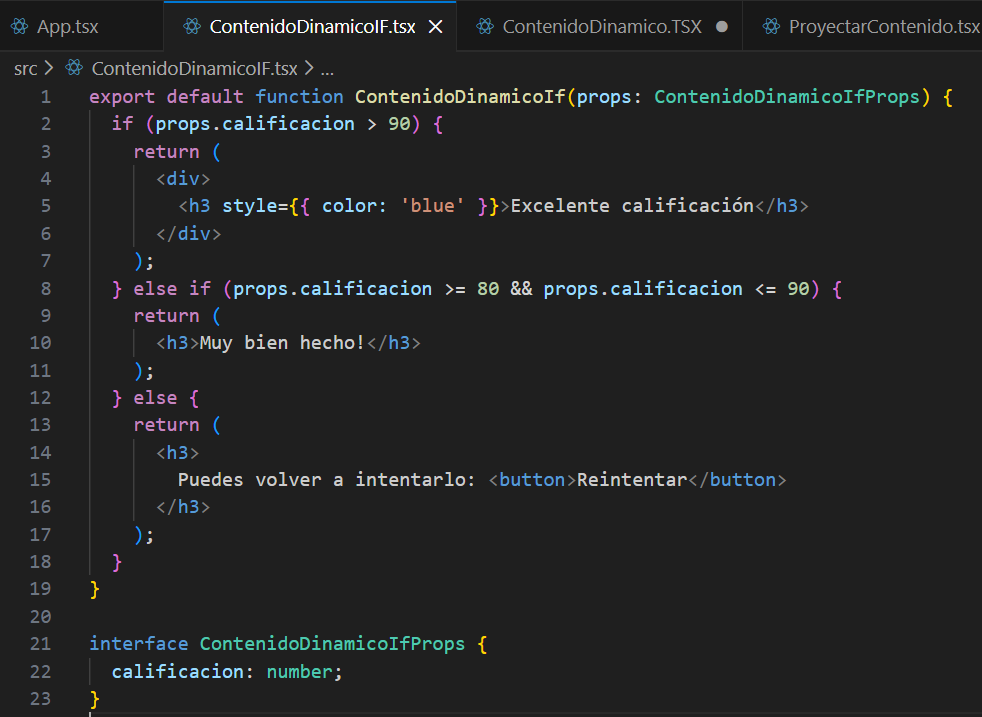
****

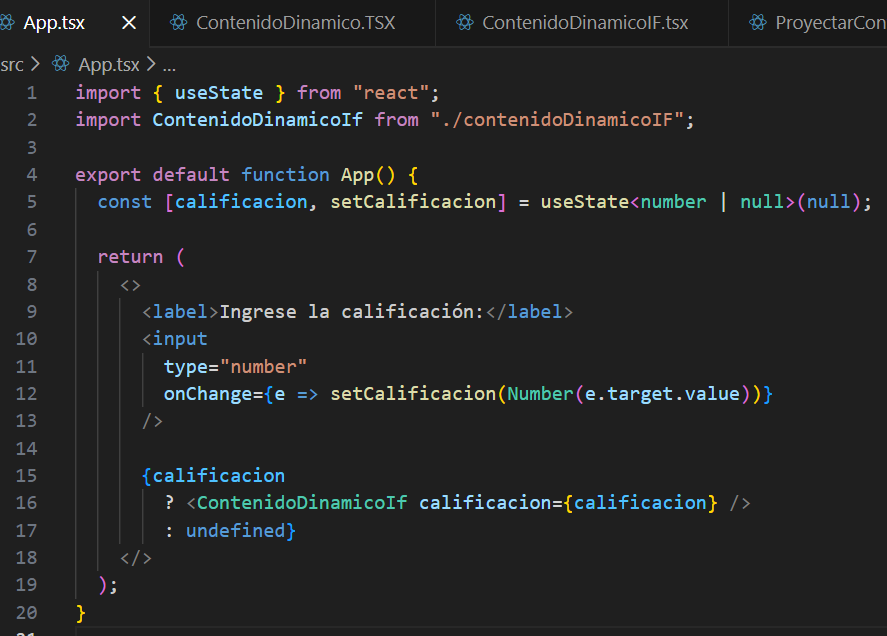
**Contenido Dinamico con operador ternario (para solo 2 opciones):**

****

****

**Contenido Dinamico con IF (2 o mas opciones):**

****

****

**Repitiendo Contenido de una tablar usando Map**

**App.tsx:**

import Tabla from "./tabla"

export default function App() {

  return (

    <>

      <Tabla />

    </>

  );

}

**Tabla.tsx:**

import React from "react";

import type Persona from "./persona.model";

export default function Tabla() {

  const personasFuente: Persona[] = [

    { id: 1, nombre: 'Felipe', departamento: 'Ingeniería' },

    { id: 2, nombre: 'Claudia', departamento: 'Recursos Humanos' },

    { id: 3, nombre: 'Roberto', departamento: 'Contabilidad' },

    { id: 4, nombre: 'Francisca', departamento: 'Contabilidad' },

    { id: 5, nombre: 'José', departamento: 'Operaciones' },

    { id: 6, nombre: 'Estephany', departamento: 'Ingeniería' },

    { id: 7, nombre: 'Norberto', departamento: 'Recursos Humanos' },

  ];

  const [personas, setPersonas] = React.useState(personasFuente);

  const removerPersona = (persona: Persona) => {

    setPersonas(personas.filter(p => p.id !== persona.id));

  }

  return (

    <table>

      <thead>

        <tr>

          <th>Nombre</th>

          <th>Departamento</th>

          <th>Acciones</th>

        </tr>

      </thead>

      <tbody>

        {personas.map(p => (

          <tr key={p.id}> 🡪 Sirve para identificar cada fila con un único valor

            <td>{p.nombre}</td>

            <td>{p.departamento}</td>

            <td>

              <button onClick={() => removerPersona(p)}>Remover</button>

            </td>

          </tr>

        ))}

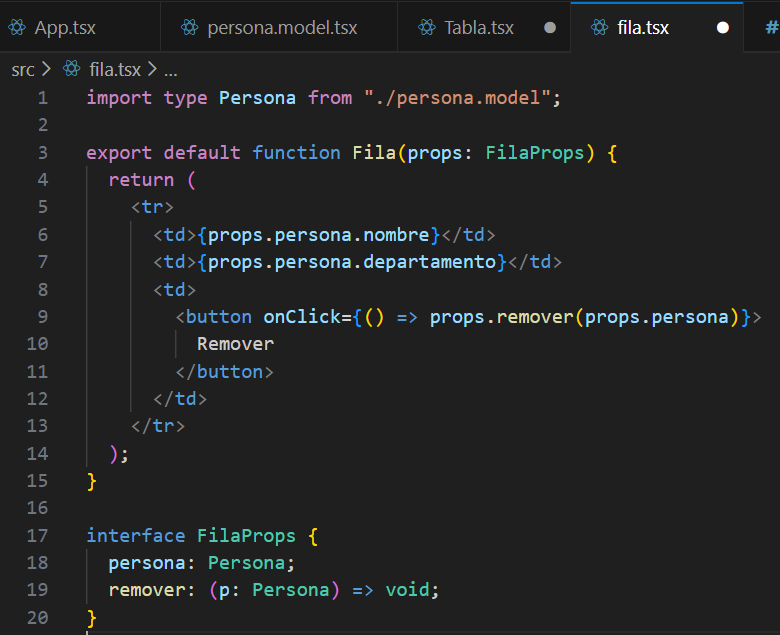
      </tbody>

    </table>

  );

}

**Comunicación Hijo a Padre (Callbacks en React):**

****

**Tabla.tsx:**

import React from "react";

import type Persona from "./persona.model";

import Fila from "./fila";

export default function Tabla() {

  const personasFuente: Persona[] = [

    { id: 1, nombre: 'Felipe', departamento: 'Ingeniería' },

    { id: 2, nombre: 'Claudia', departamento: 'Recursos Humanos' },

    { id: 3, nombre: 'Roberto', departamento: 'Contabilidad' },

    { id: 4, nombre: 'Francisca', departamento: 'Contabilidad' },

    { id: 5, nombre: 'José', departamento: 'Operaciones' },

    { id: 6, nombre: 'Estephany', departamento: 'Ingeniería' },

    { id: 7, nombre: 'Norberto', departamento: 'Recursos Humanos' },

  ];

  const [personas, setPersonas] = React.useState(personasFuente);

  const removerPersona = (persona: Persona) => {

    setPersonas(personas.filter(p => p.id !== persona.id));

  }

  return (

    <table>

      <thead>

        <tr>

          <th>Nombre</th>

          <th>Departamento</th>

          <th>Acciones</th>

        </tr>

      </thead>

      <tbody>

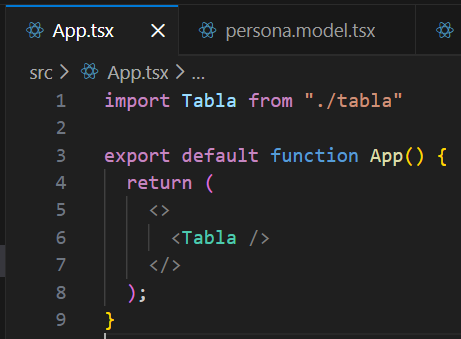
        {personas.map(p => <Fila key={p.id} persona={p} remover={removerPersona} />)}

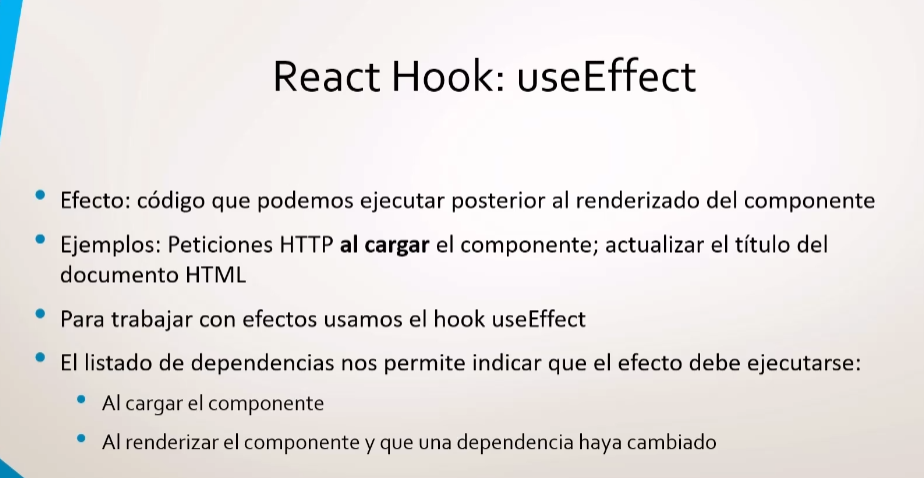
      </tbody>

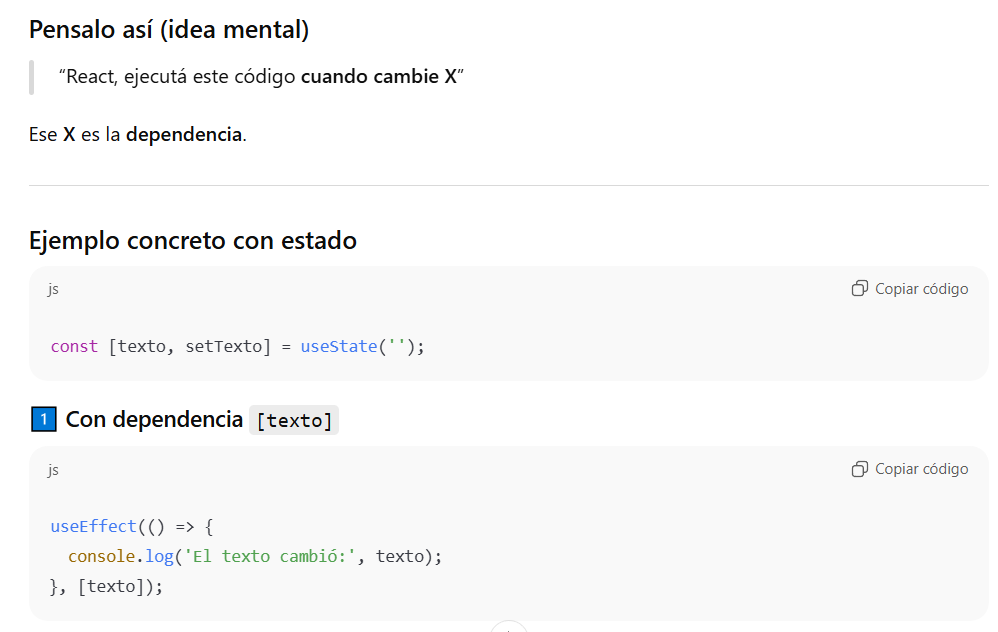
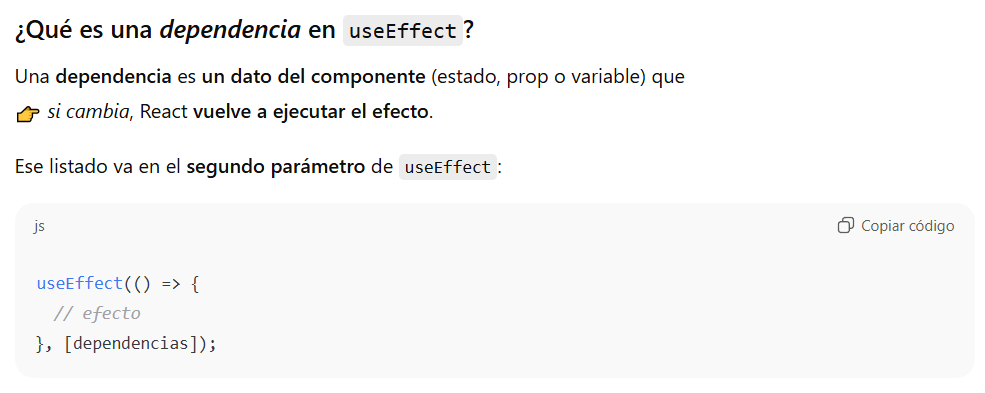
    </table>

  );

}

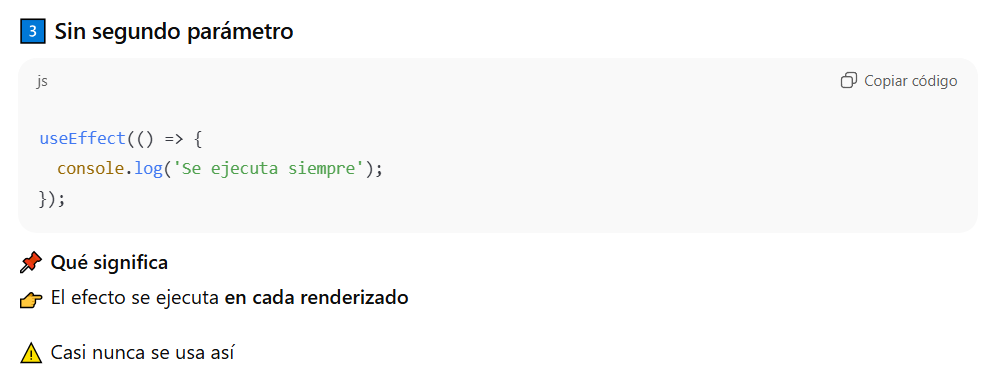
****

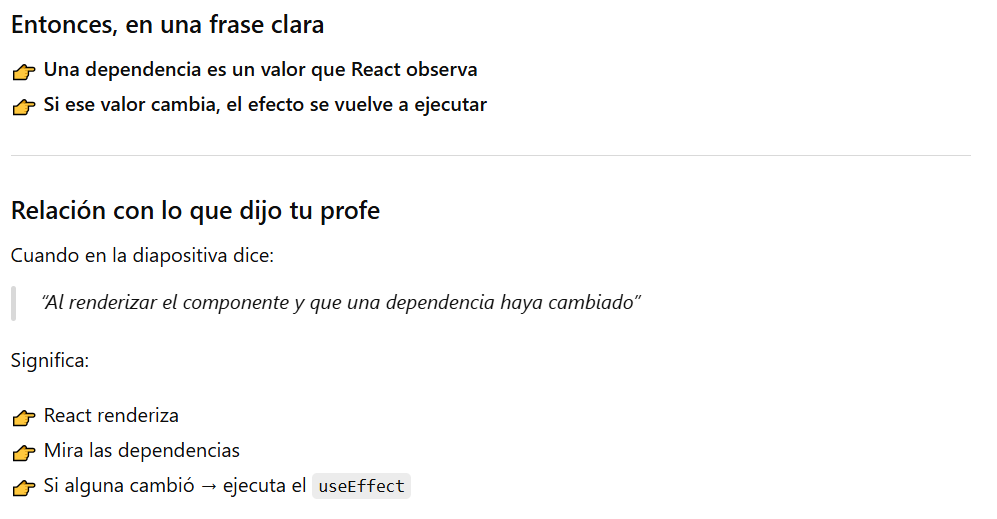
****

****

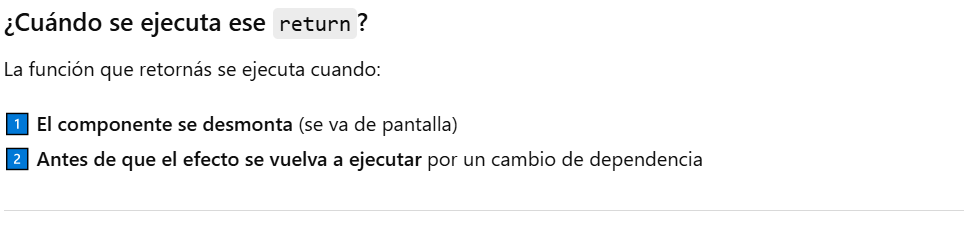
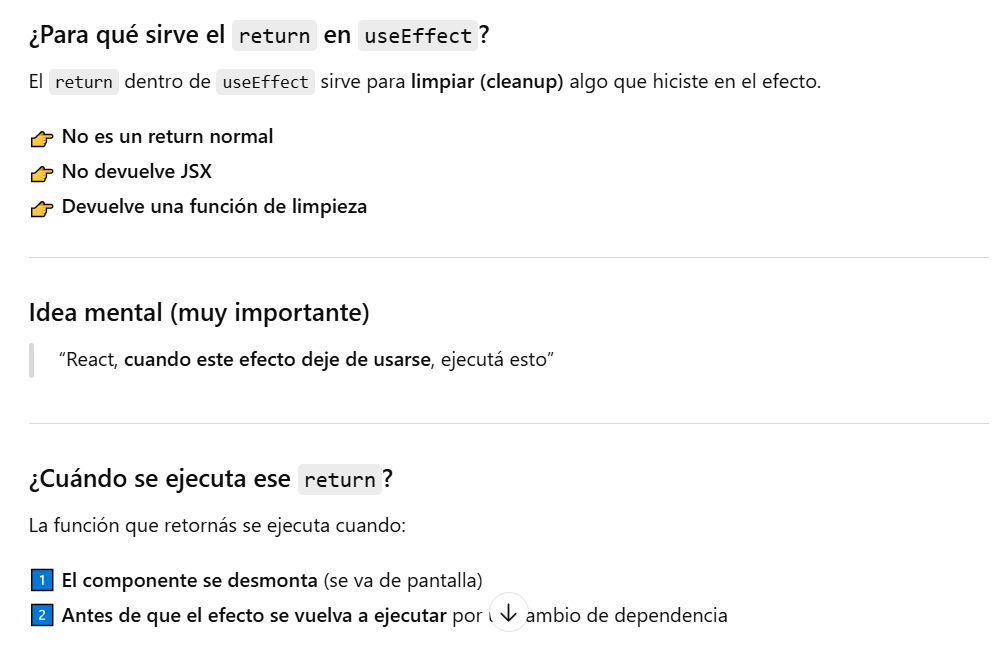
****

****

****

****

****

****

**Ejemplo:**

**EjemploUseEffect.tsx:**

import { useEffect, useState } from "react";

export default function EjemploUseEffect() {

  const [clicks, setClicks] = useState(0);

  const [hora, setHora] = useState(new Date());

  /\* useEffect con [] indica que el efecto se jecuta una sola vez al montar el componente \*/

  useEffect(() => {

    console.log('el componente ha cargado');

    return () => console.log('desmontando el componente');

  }, []);

  /\* useEffect con [clicks] indica que se ejecuta cada vez que cambia 'clicks' \*/

  useEffect(() => {

    console.log(`hook del clic`);

    document.title = `${clicks} veces`;

  }, [clicks]);

  /\* useEffect sin dependencias indica que el efecto se ejecuta en cada renderizado del componente \*/

  useEffect(() => {

    const timeId = setInterval(() => {

        setHora(new Date());

    }, 1000);

    return () => clearInterval(timeId); // Limpieza del intervalo al desmontar

  },);

  return (

    <>

      <h2>Ejemplo UseEffect</h2>

      <div>

        <button onClick={() => setClicks(clicks + 1)}>

          Me has clickeado {clicks} veces

        </button>

      </div>

      <div>

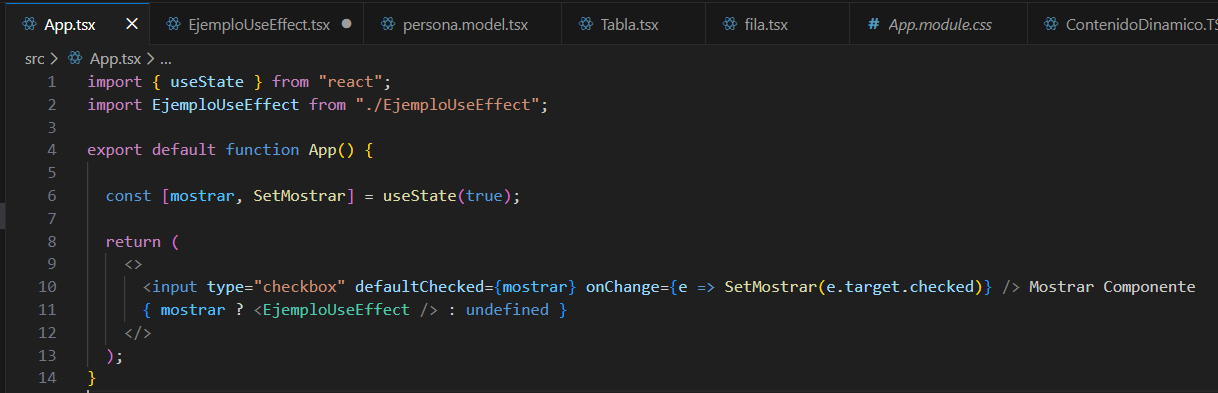
        La hora actual es {hora.toTimeString()}

      </div>

    </>

  );

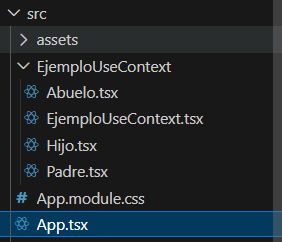
}

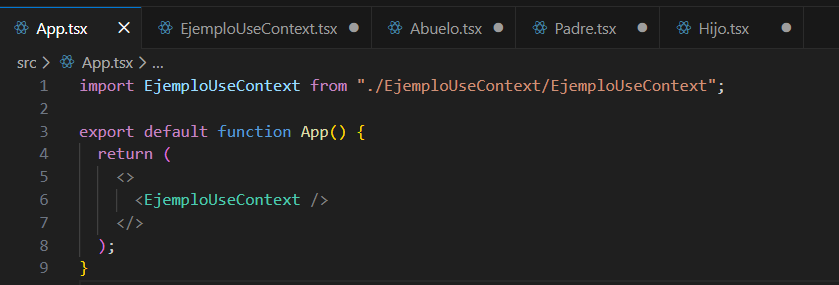
****

**React Hooks: useContext**

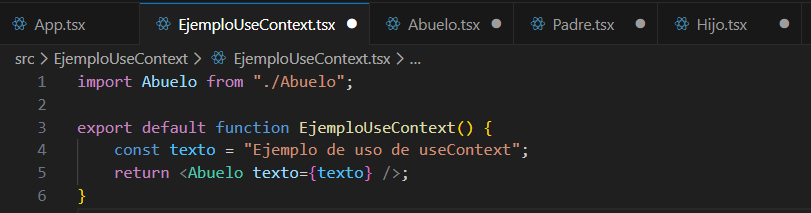
**useContext** permite que un componente hijo acceda a datos compartidos sin usar props.  
El **Provider** es el componente **padre** que envuelve a sus hijos y provee un valor mediante value.  
Los **componentes hijos** consumen ese valor usando useContext, siempre que estén dentro del Provider.

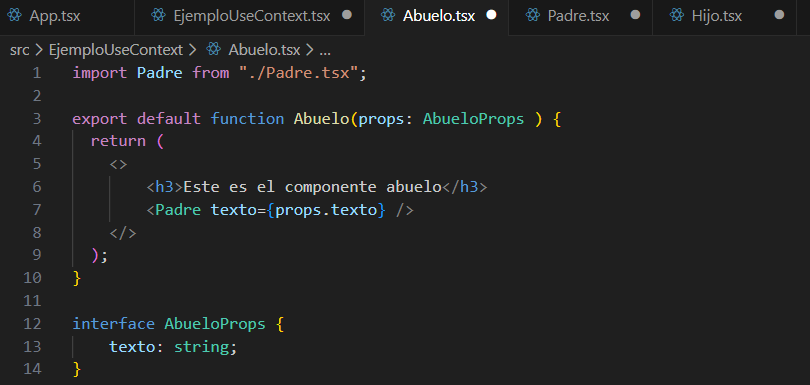
Supangamos que queremos mandar una variable definida en EjemploUseContext.tsx a Hijo.tsx:

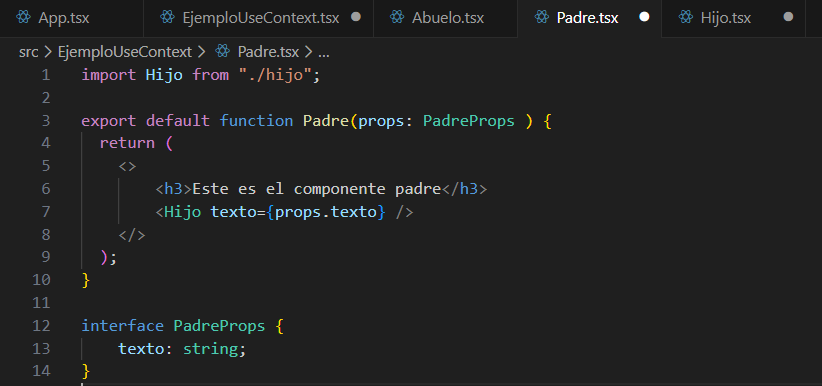


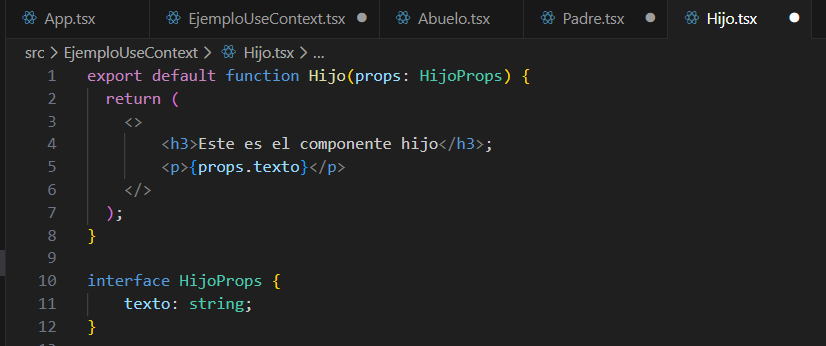


Normalmente lo haríamos de esta forma:

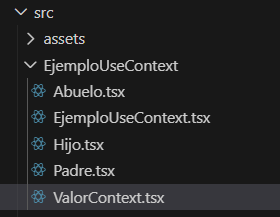


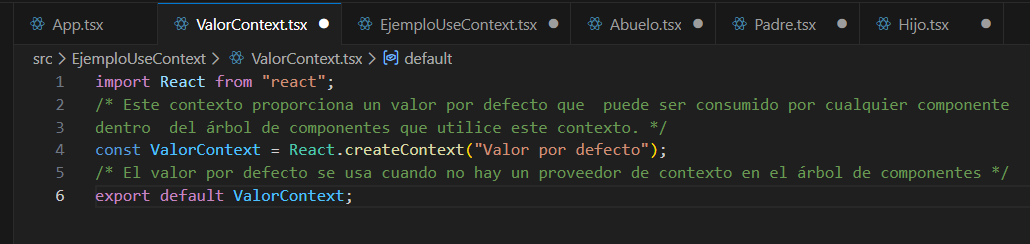


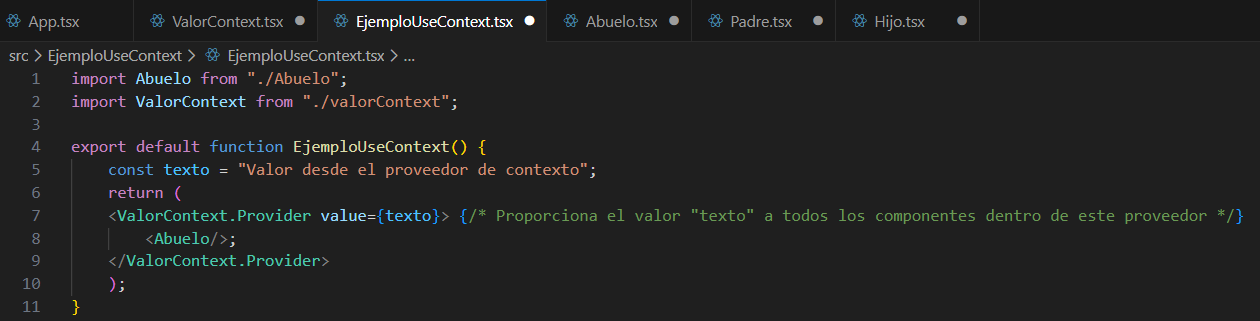


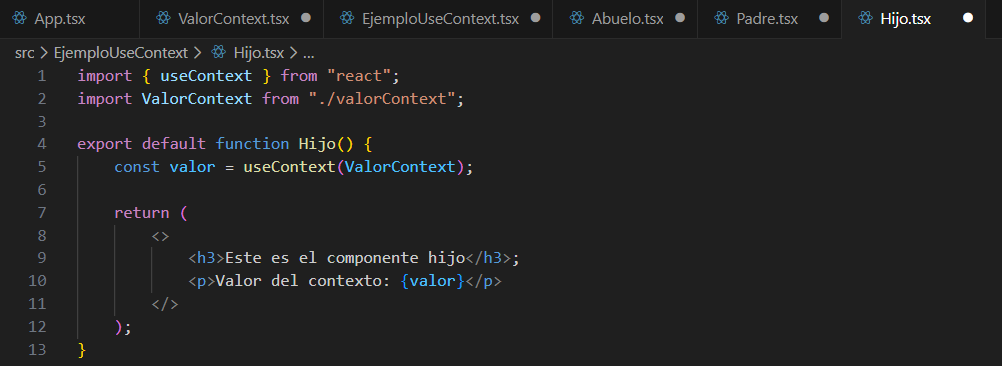


Todo esto es muy tedioso y la mejor manera de resolverlo es con **UseContext**:





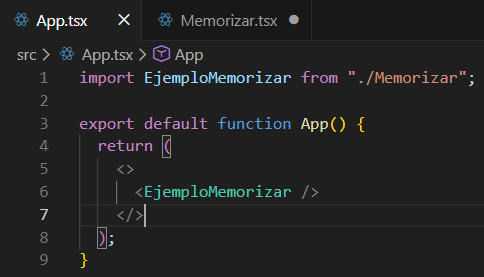
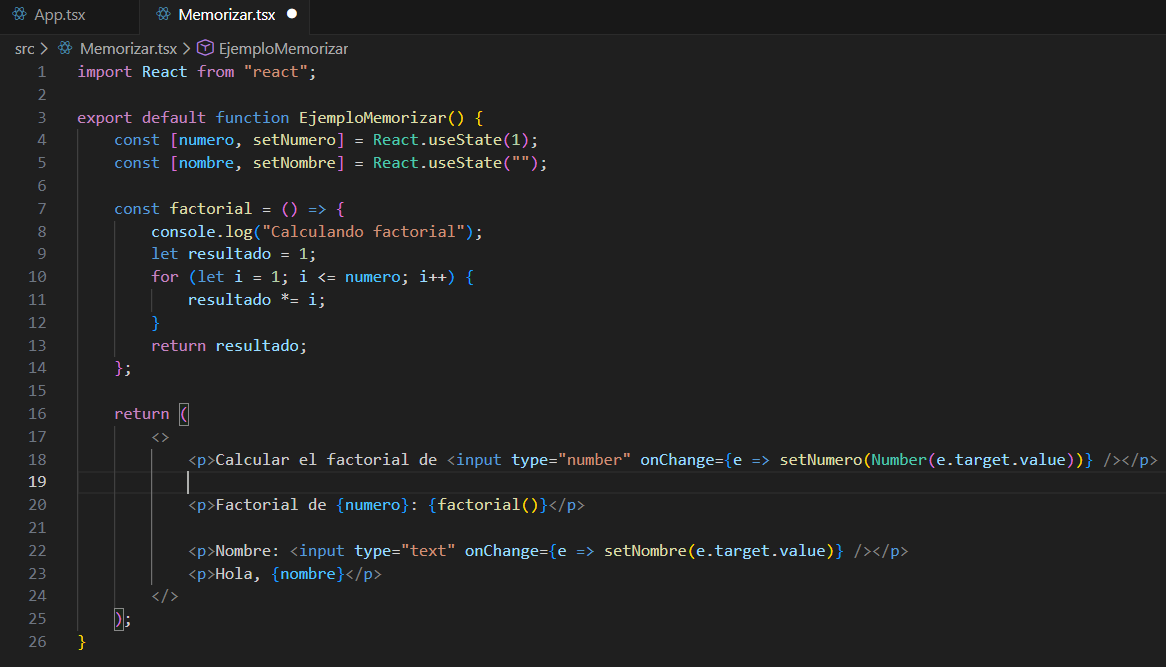


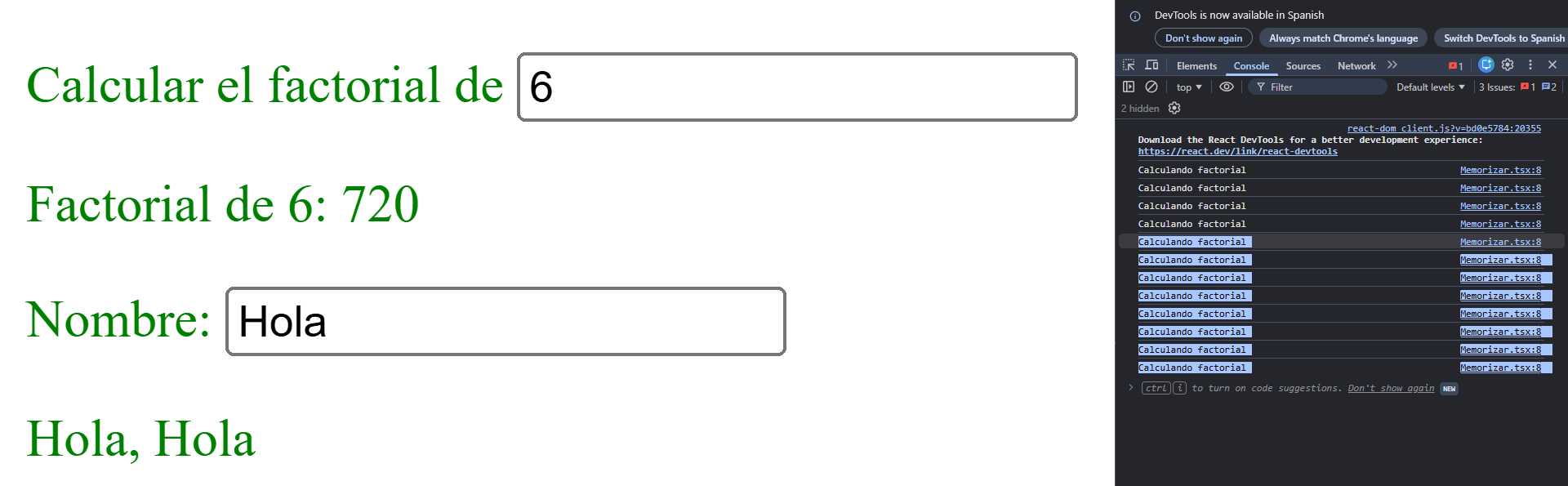


**React Hooks: useMemo**

**useMemo** guarda el resultado de un cálculo y solo lo recalcula si cambian sus dependencias.

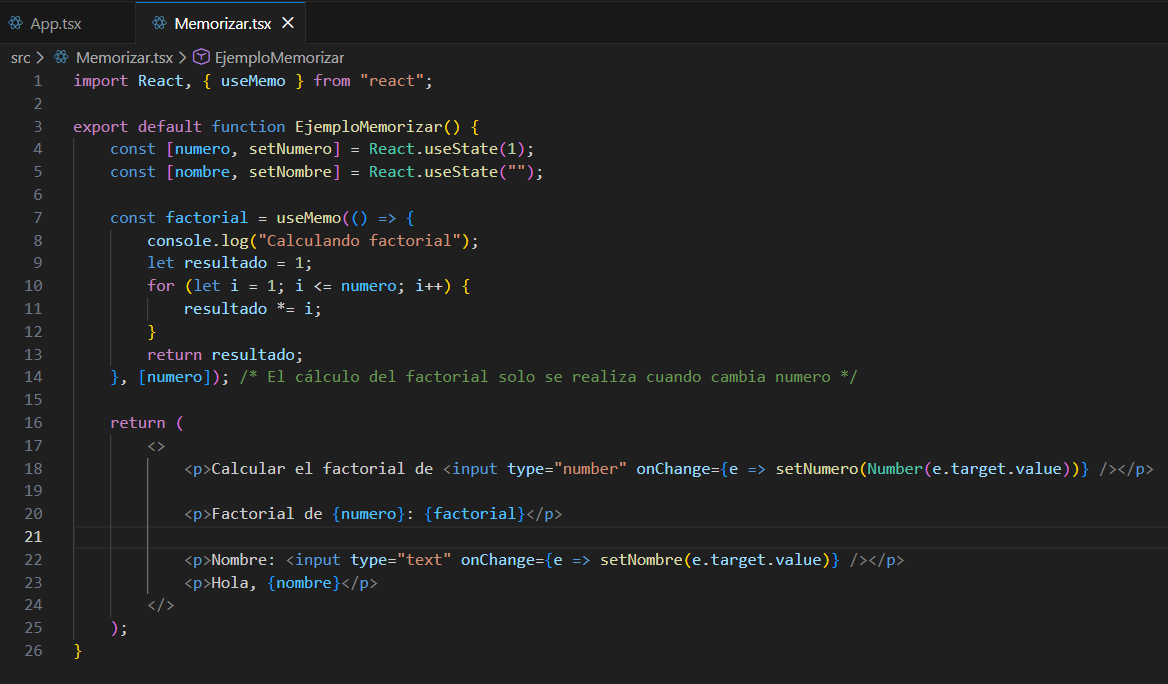
Problemática:

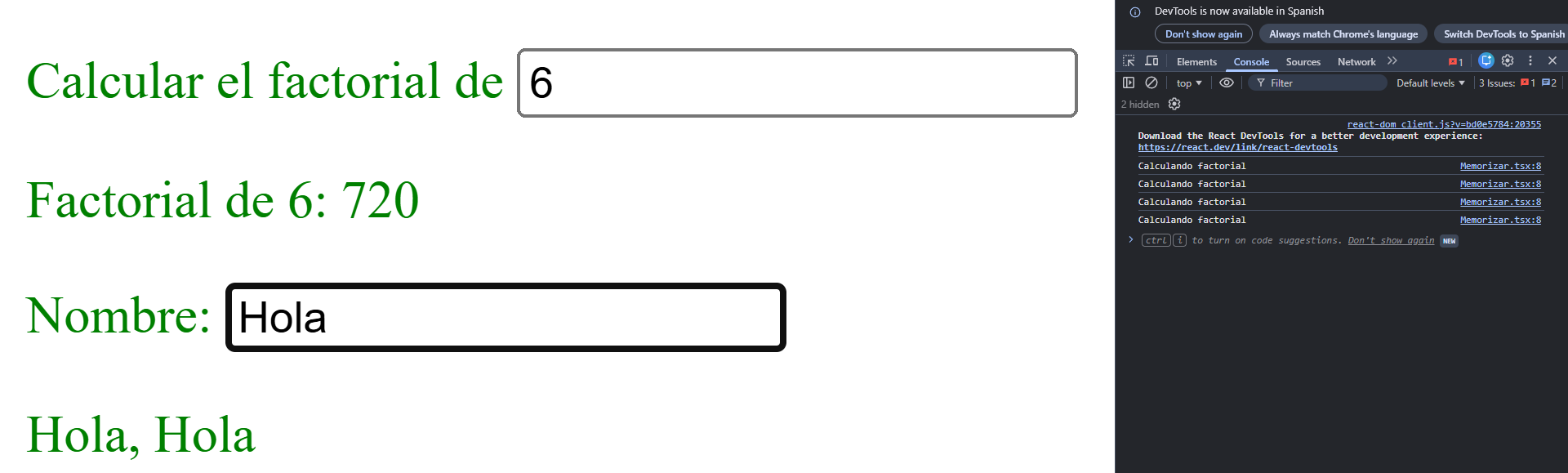
 

El estado nombre al modificarse con cada valor que se ingresa por teclado react renderiza, al hacer esto inevitavlemente se vuelve a calcular factorial() para el estado numero que se mantiene entre renderizados.

Esto es un problema en casos de que existan funciones que hagan cálculos muy pesados, que al volverse a repetir contantemente me ralentiza el sistema. Solucón 🡪 **useMemo**.

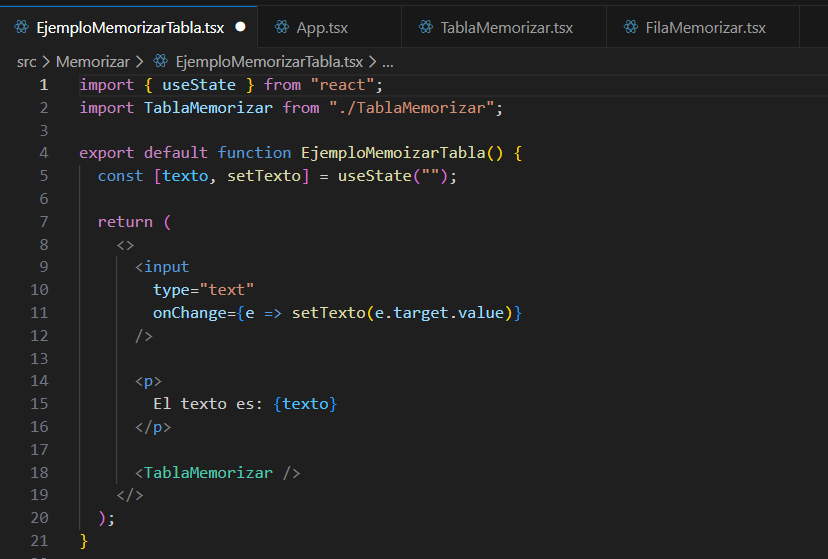
Con use**Memo**:





*Solucionado, la función factorial expresada solo se ejecuta si numero cambia.*

Otro problema que soluciona useMemo:

**

*TablaMemorizar.tsx:*

import React from "react";

import type Persona from "../persona.model";

import FilaMemorizar from "./FilaMemorizar";

export default function TablaMemorizar() {

  console.log("Se renderiza la tabla");

  const personasFuente: Persona[] = [

    { id: 1, nombre: 'Felipe', departamento: 'Ingeniería' },

    { id: 2, nombre: 'Claudia', departamento: 'Recursos Humanos' },

    { id: 3, nombre: 'Roberto', departamento: 'Contabilidad' },

    { id: 4, nombre: 'Francisca', departamento: 'Contabilidad' },

    { id: 5, nombre: 'José', departamento: 'Operaciones' },

    { id: 6, nombre: 'Estephany', departamento: 'Ingeniería' },

    { id: 7, nombre: 'Norberto', departamento: 'Recursos Humanos' },

  ];

  const [personas, setPersonas] = React.useState(personasFuente);

  const removerPersona = (persona: Persona) => {

    setPersonas(personas.filter(p => p.id !== persona.id));

  }

  return (

    <table>

      <thead>

        <tr>

          <th>Nombre</th>

          <th>Departamento</th>

          <th>Acciones</th>

        </tr>

      </thead>

      <tbody>

        {personas.map(p => <FilaMemorizar key={p.id} persona={p} remover={removerPersona} />)}

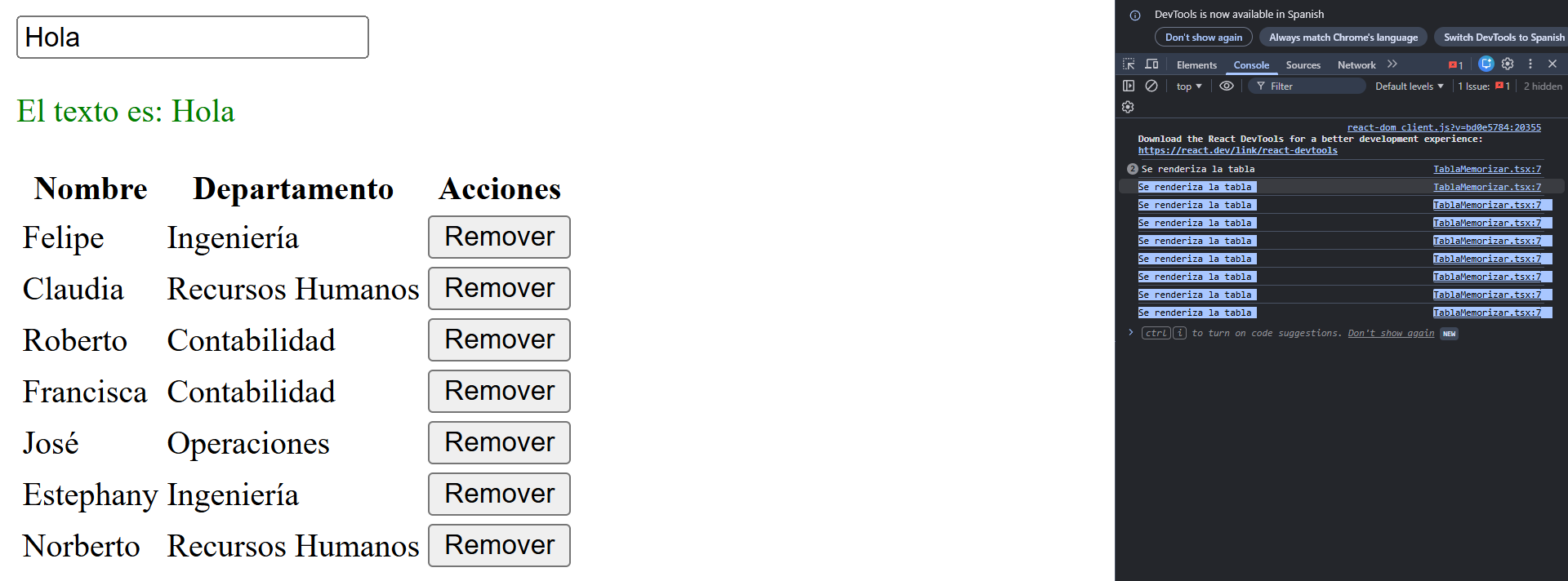
      </tbody>

    </table>

  );

}

*Cada vez que modifico texto de EjemploMemorizarTabla.txt, esot renderizando de nuevo el componente TablaMemorizar.*



Por lo tanto la lógica es igual que antes, puedo usar React.memo(componente), en vez de UseMemo, para memorizar el componente y no renderizarlo si se mantiene igual, esto es factible ya que un componente no es mas que una función al igual que factorial(). Por lo tanto, el componente TablaMemorizar quedaría asi:

import React, { memo } from "react";

import type Persona from "../persona.model";

import FilaMemorizar from "./FilaMemorizar";

const TablaMemorizar = memo(function TablaMemorizar() {

  console.log("Se renderiza la tabla");

  const personasFuente: Persona[] = [

    { id: 1, nombre: 'Felipe', departamento: 'Ingeniería' },

    { id: 2, nombre: 'Claudia', departamento: 'Recursos Humanos' },

    { id: 3, nombre: 'Roberto', departamento: 'Contabilidad' },

    { id: 4, nombre: 'Francisca', departamento: 'Contabilidad' },

    { id: 5, nombre: 'José', departamento: 'Operaciones' },

    { id: 6, nombre: 'Estephany', departamento: 'Ingeniería' },

    { id: 7, nombre: 'Norberto', departamento: 'Recursos Humanos' },

  ];

  const [personas, setPersonas] = React.useState(personasFuente);

  const removerPersona = (persona: Persona) => {

    setPersonas(personas.filter(p => p.id !== persona.id));

  }

  return (

    <table>

      <thead>

        <tr>

          <th>Nombre</th>

          <th>Departamento</th>

          <th>Acciones</th>

        </tr>

      </thead>

      <tbody>

        {personas.map(p => <FilaMemorizar key={p.id} persona={p} remover={removerPersona} />)}

      </tbody>

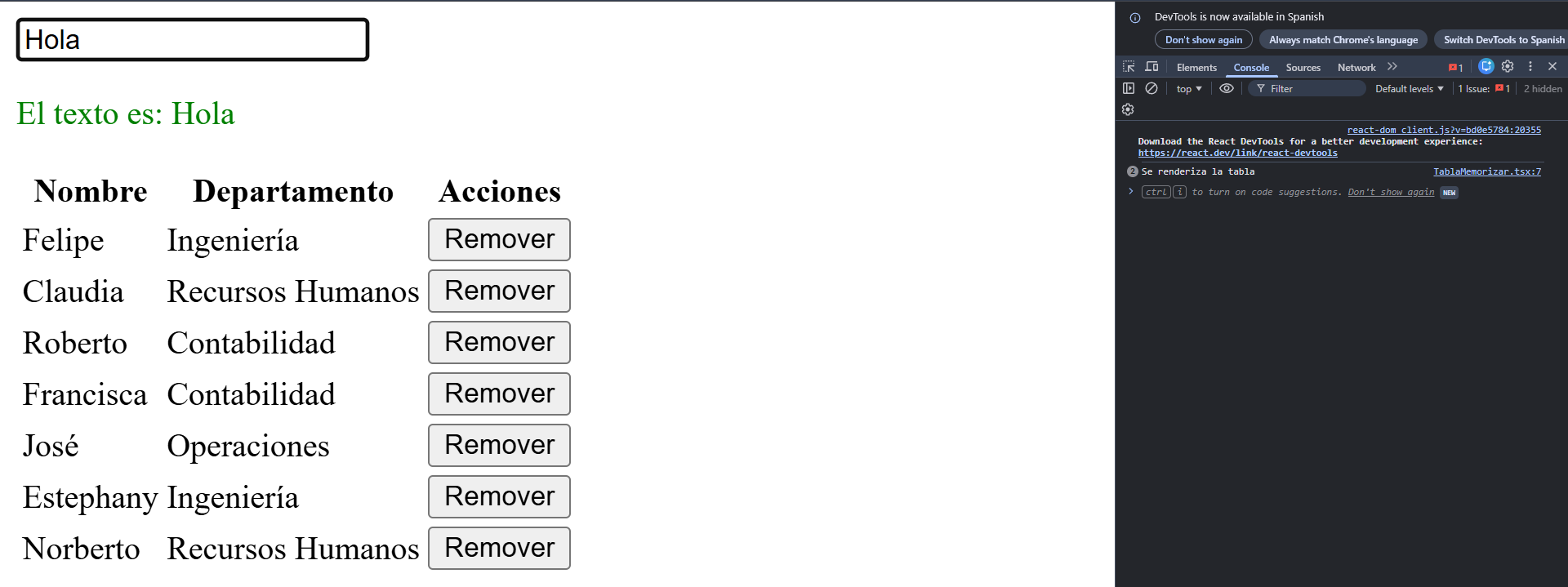
    </table>

  );

});

export default TablaMemorizar;

Ya no se renderiza la Tabla:



Si llegamos a borrar una fila obviamente si se va a renderizar.

### Resumen mental

* useMemo → memoriza valores
* React.memo → memoriza componentes

**React Hooks: useCallback**

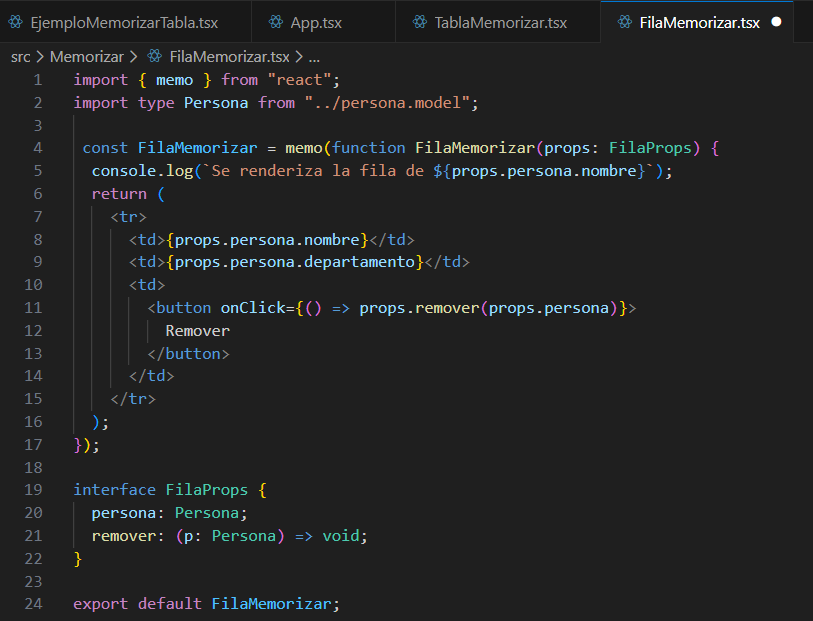
El componente TablaMemorizar.tsx:

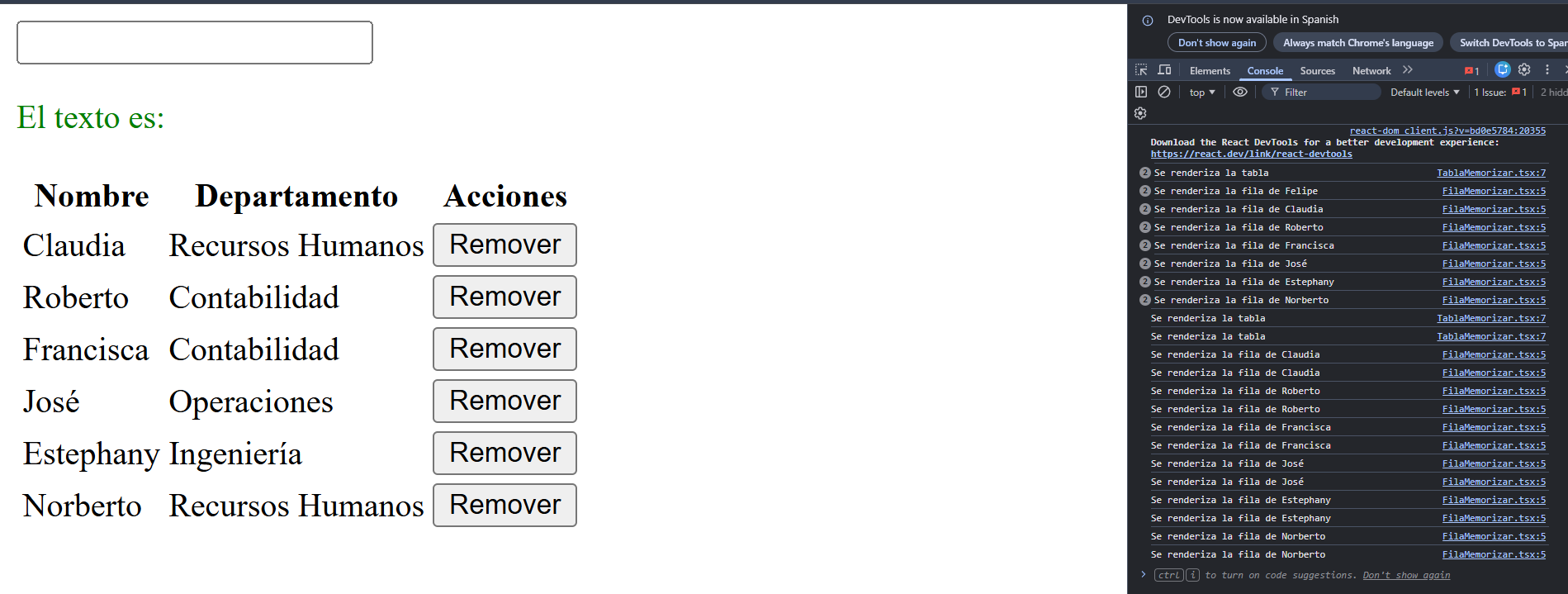


Es padre del componente FilaMemorizar.tsx:

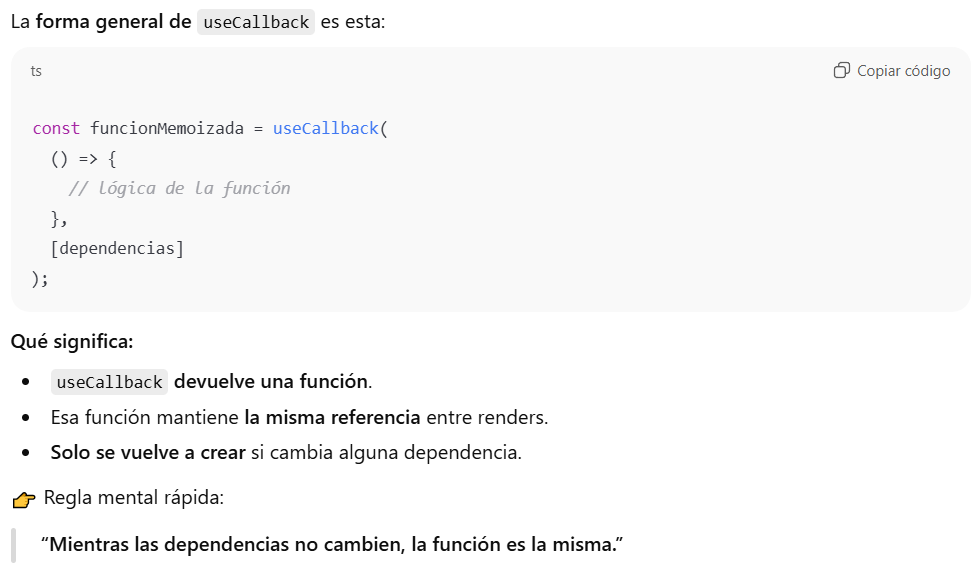


Notar que aunque use React.memo() igual no logro hacer que se dejen de renderizar todas las filas:

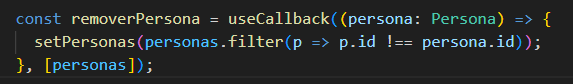


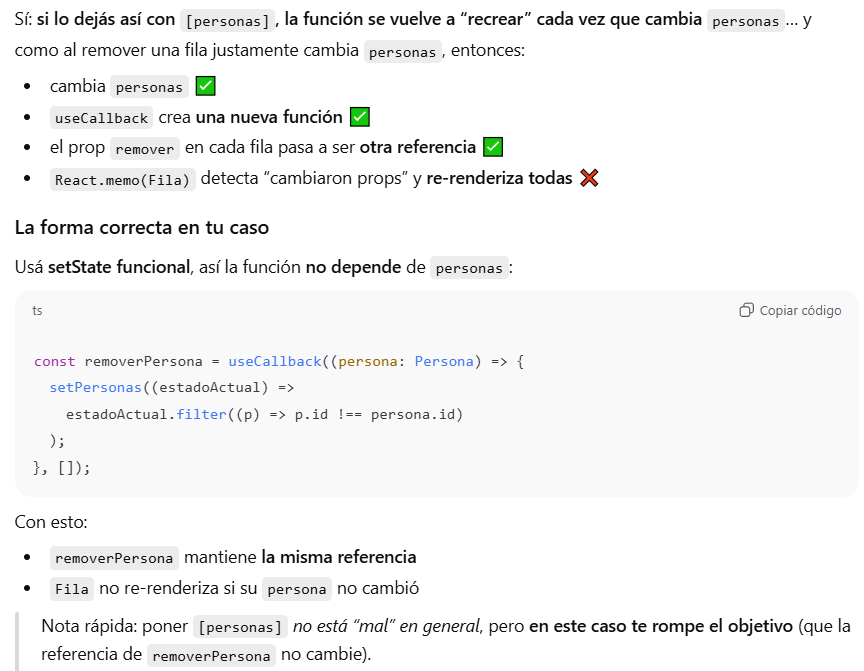


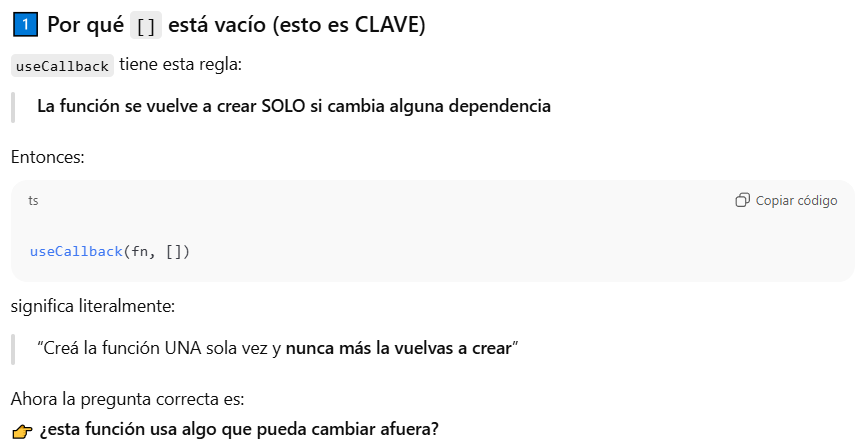
**memo() solo no sirve** porque aunque el componente hijo esté memorizado, **sus props cambian igual** si desde el componente padre se le pasa una función creada en cada render (remover). En JavaScript, **una función es un objeto por referencia**, y cada vez que el padre se renderiza React **crea una nueva función**, o sea **una referencia distinta**, aunque haga exactamente lo mismo. Entonces React.memo detecta que el prop cambió (referencia A ≠ referencia B) y vuelve a renderizar el componente. useCallback soluciona esto **manteniendo la misma referencia de la función entre renders**, permitiendo que memo() compare props iguales y realmente evite renderizados innecesarios.

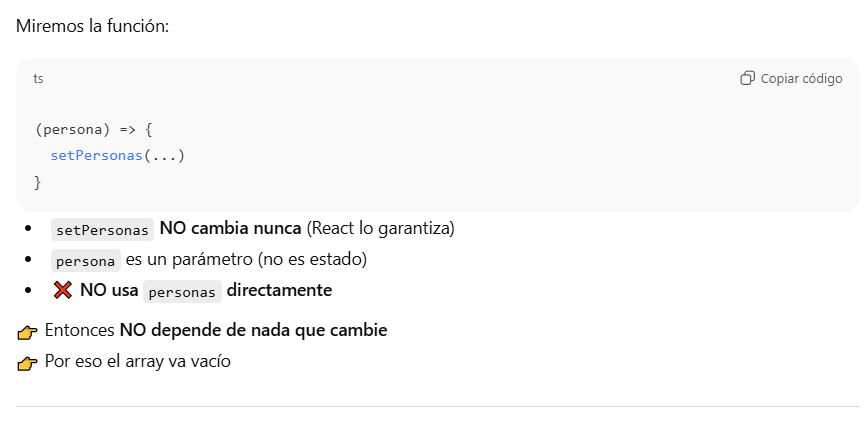


Debido a esto yo NO puedo hacer esto:

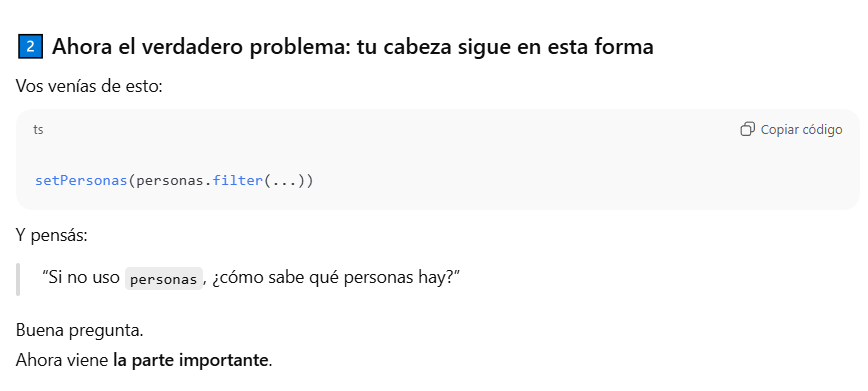


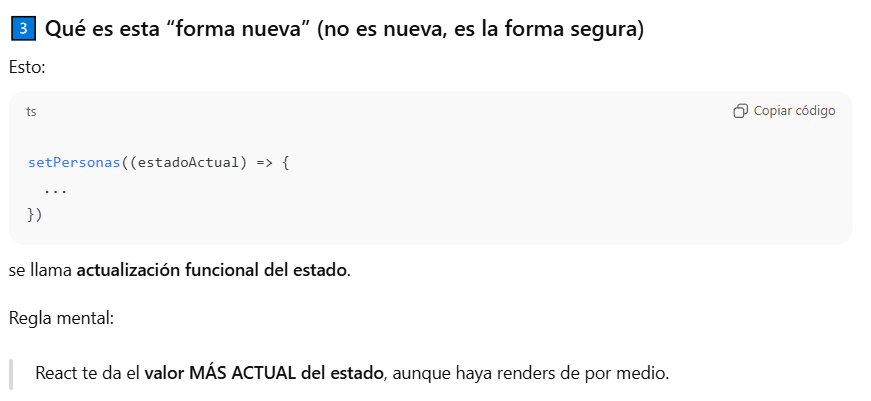


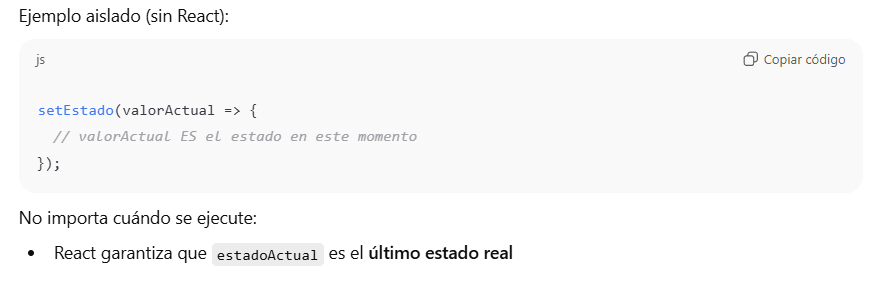




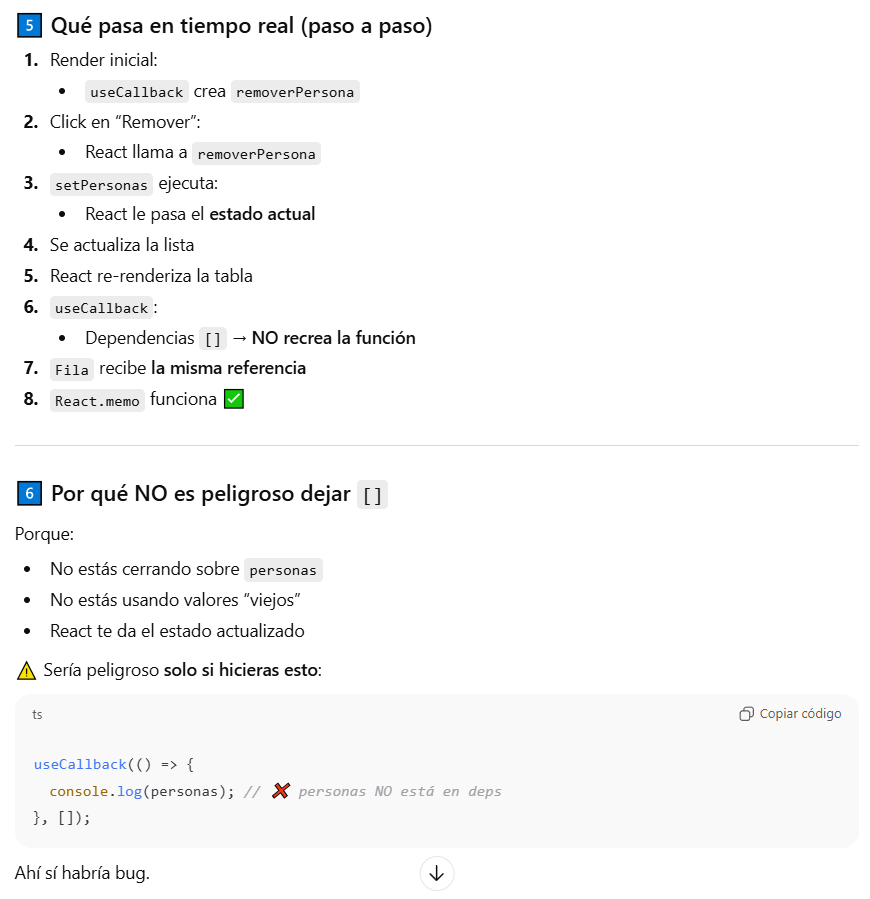
**¡¡¡ DE ACÁ PARA ADELANTE NO ENTENDÍ NADA ! ! !**

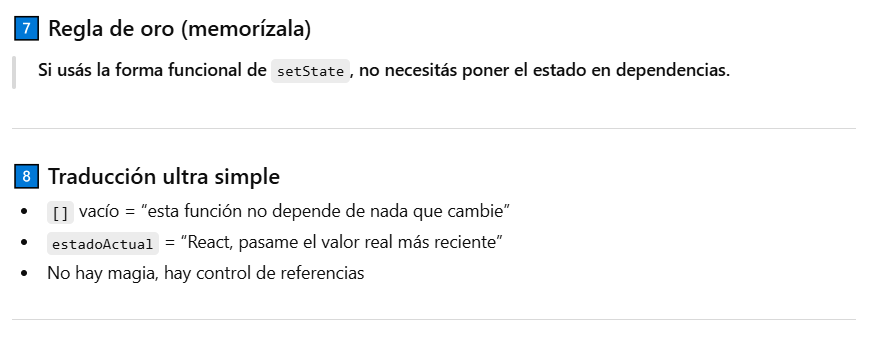




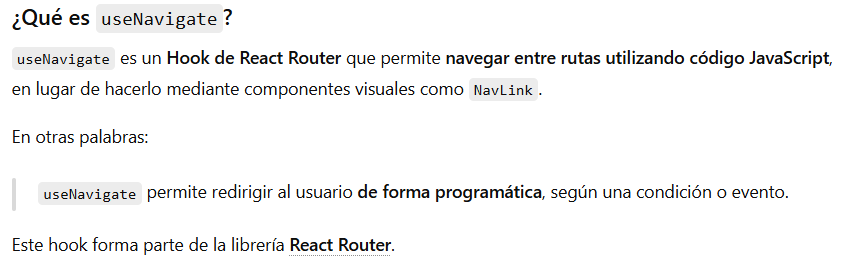


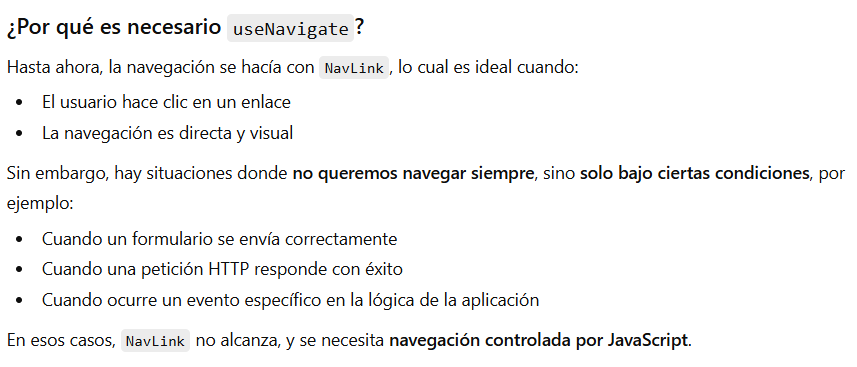


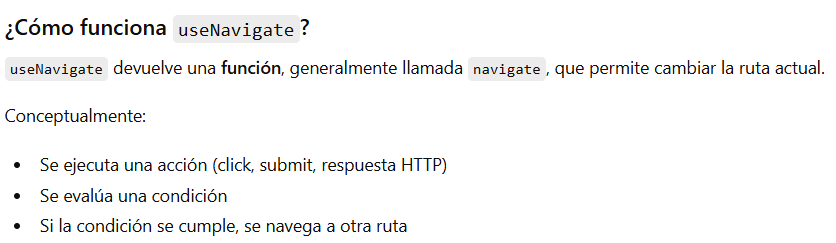




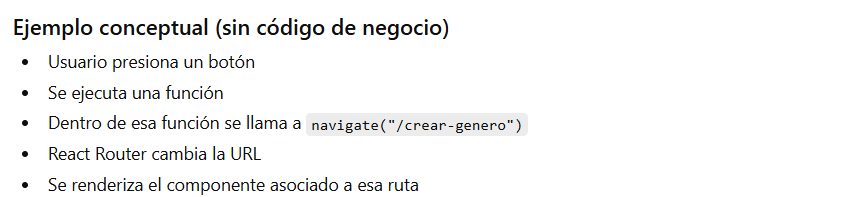
**React Hook: useNavigate**

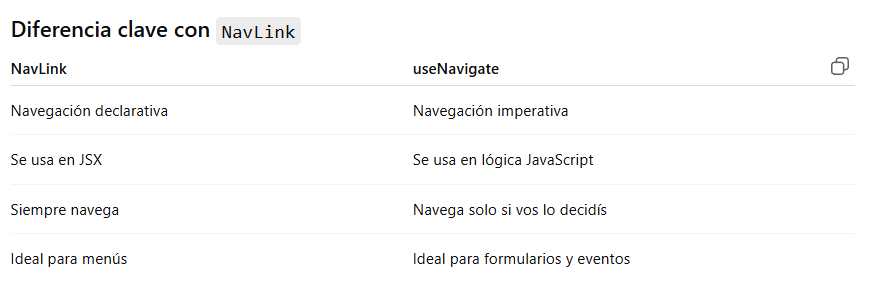
****

****

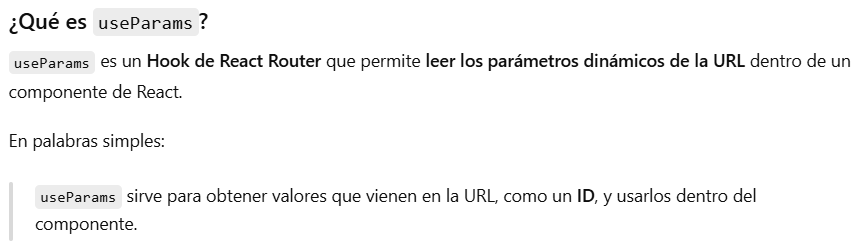
****

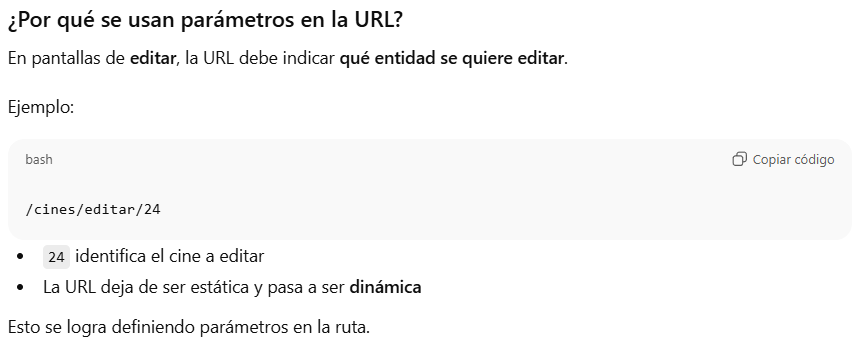
****

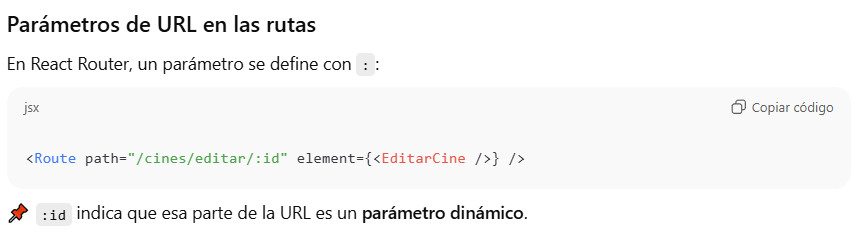
****

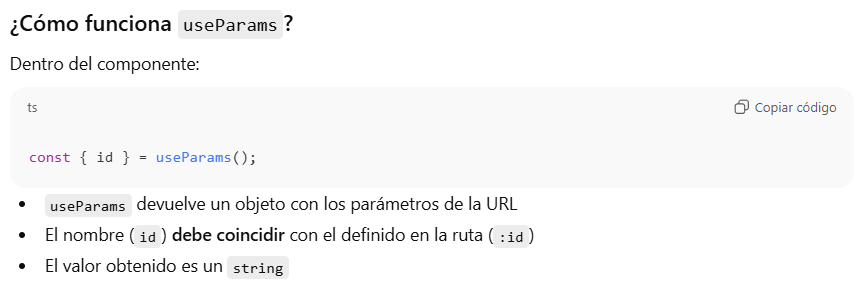
****

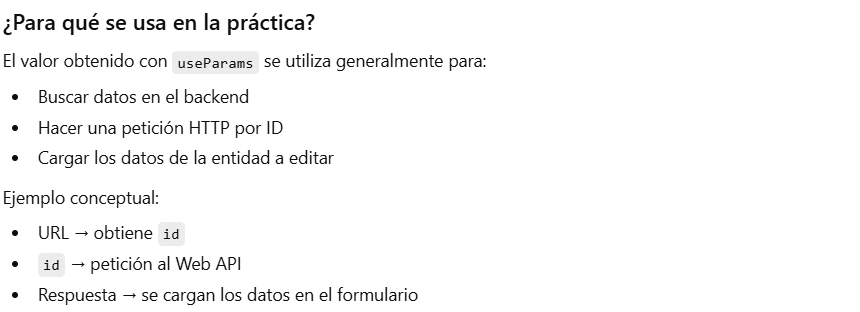
**React Hook: useParams**

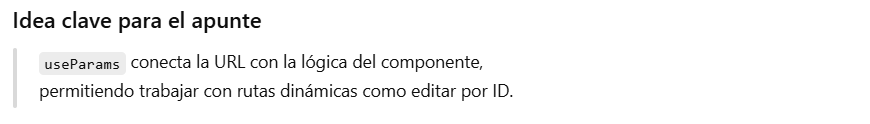
****

****

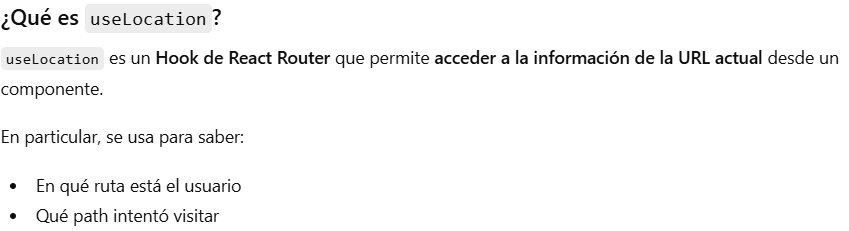
****

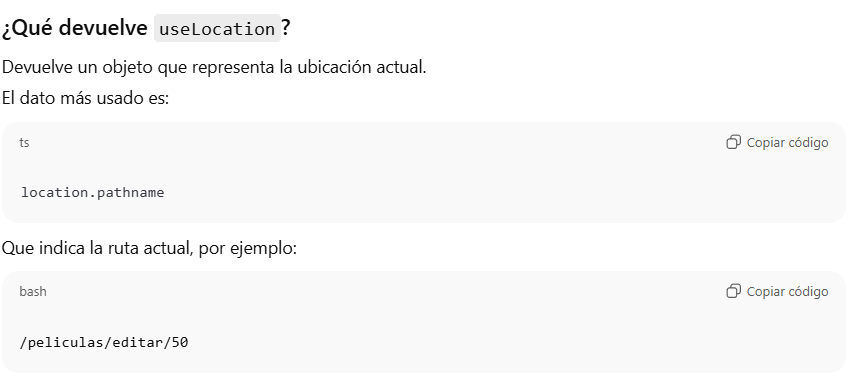
****

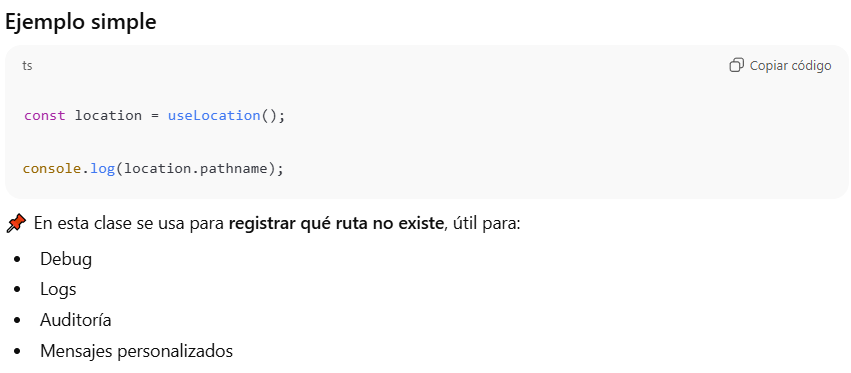
****

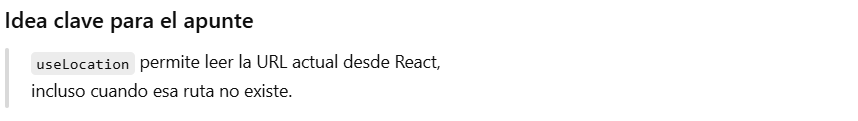
****

**React Hook: useLocation**

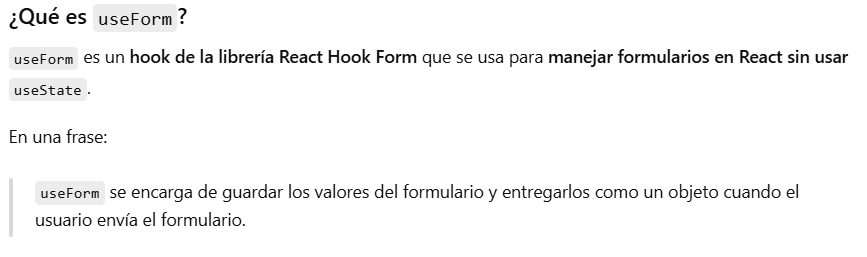
****

****

****

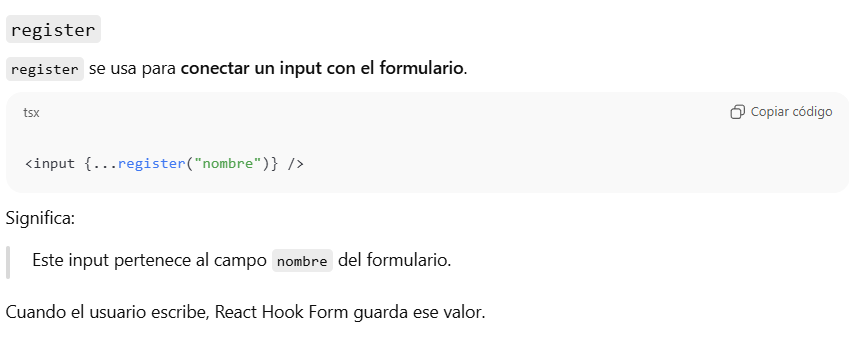
****

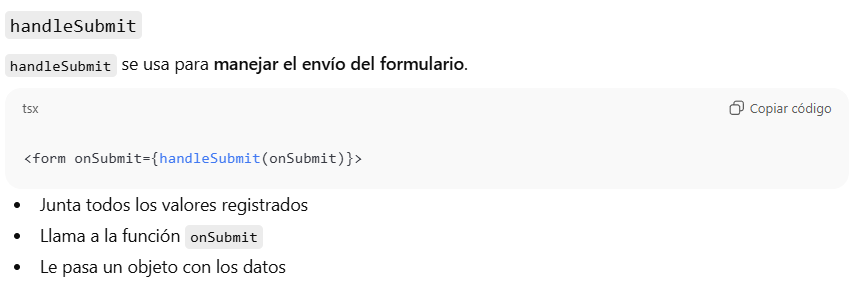
**React Hook: useForm (React Hook Form)**

****

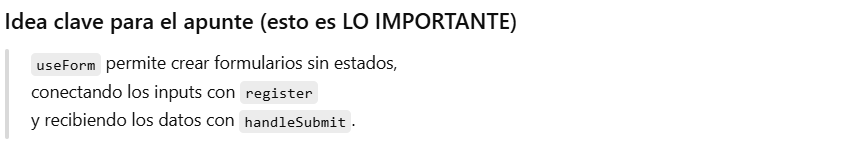
****

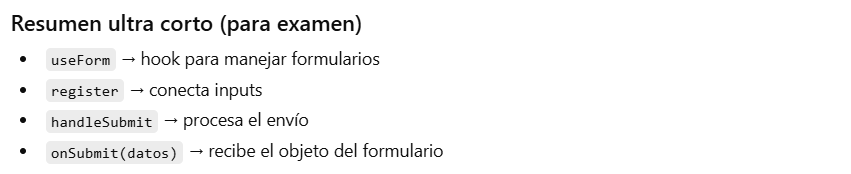
****

****

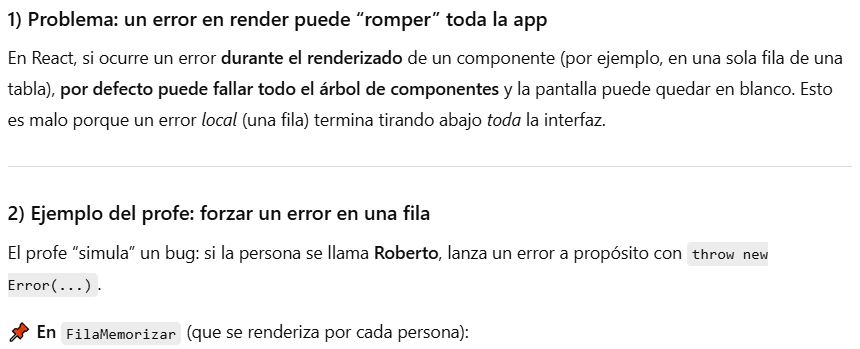
****

****

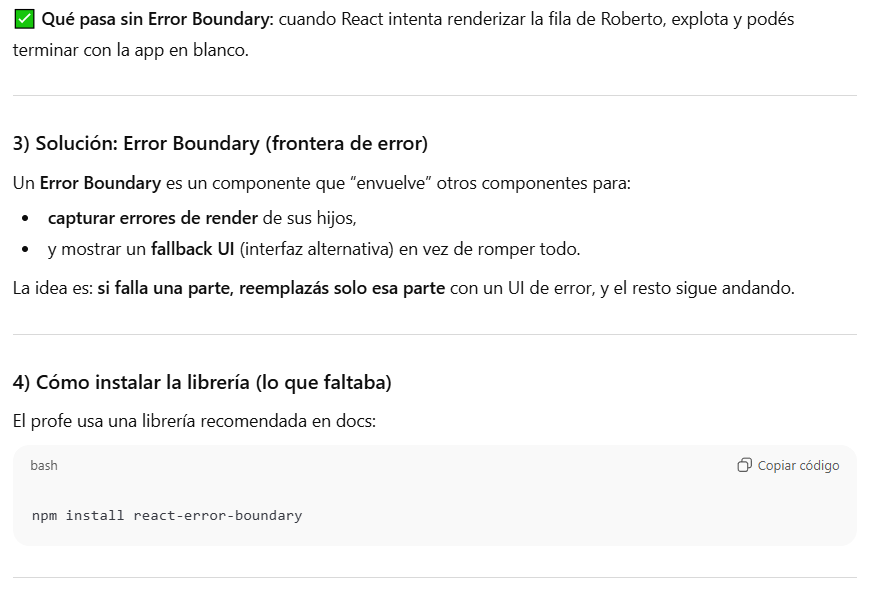
****

****

**Manejando errores con Error Boundaries**

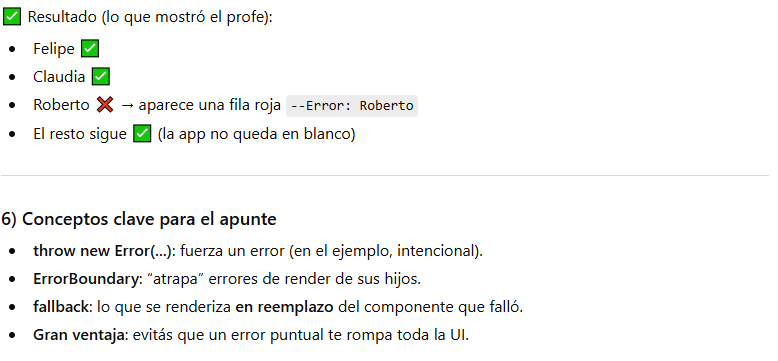






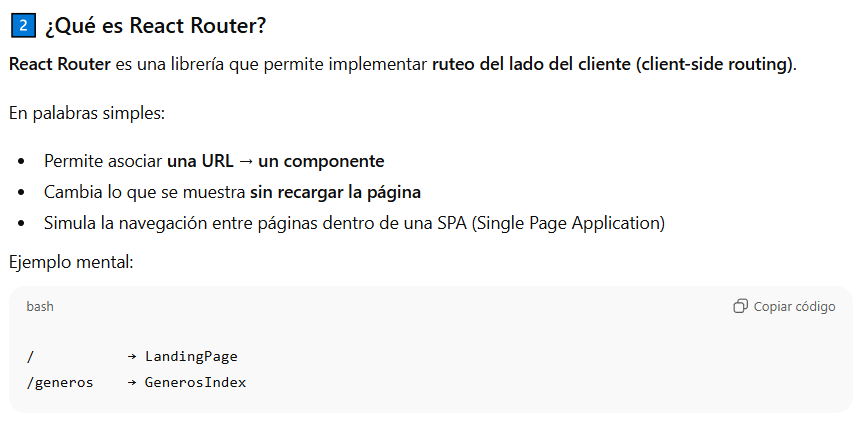


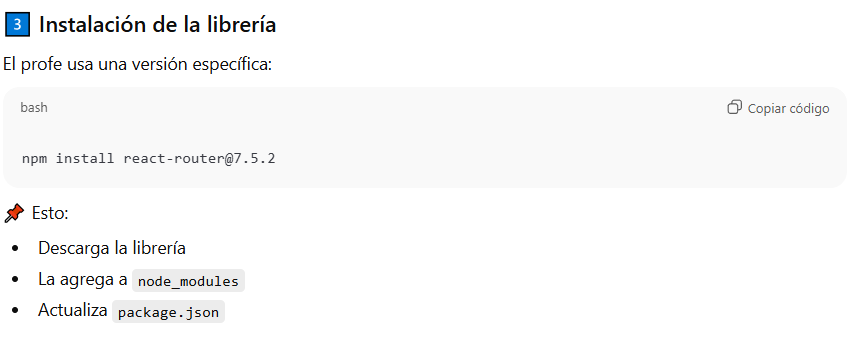


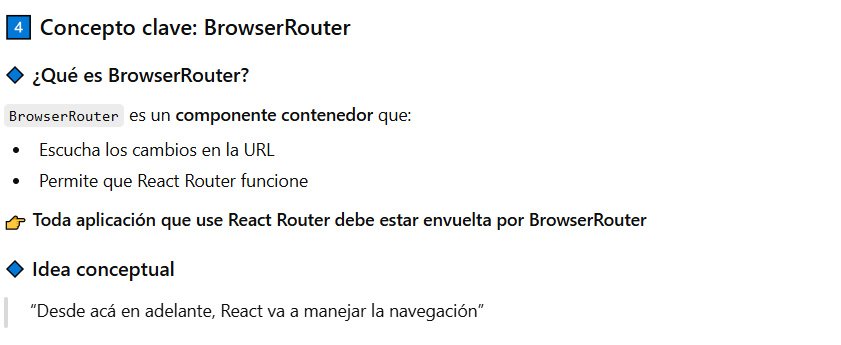


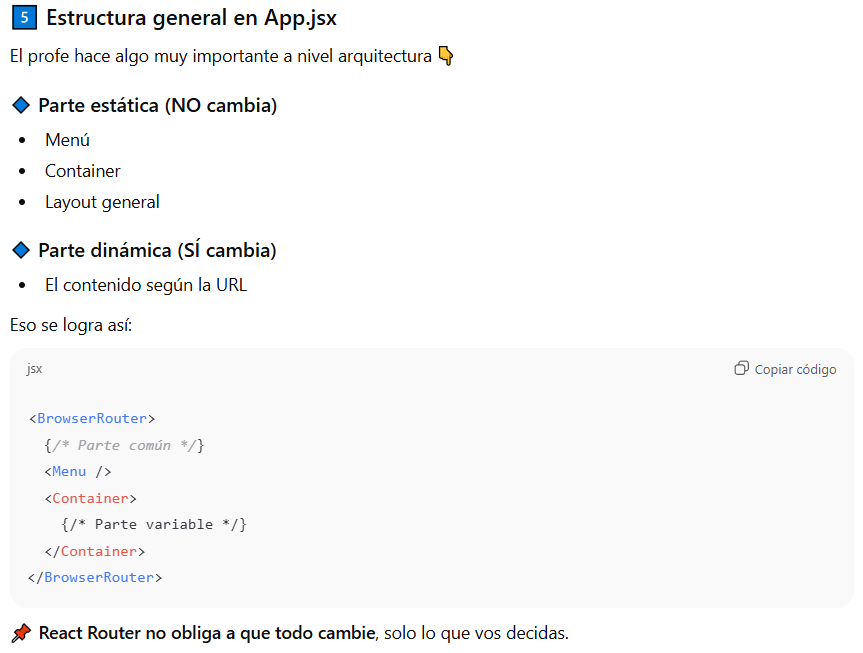
**React Router**

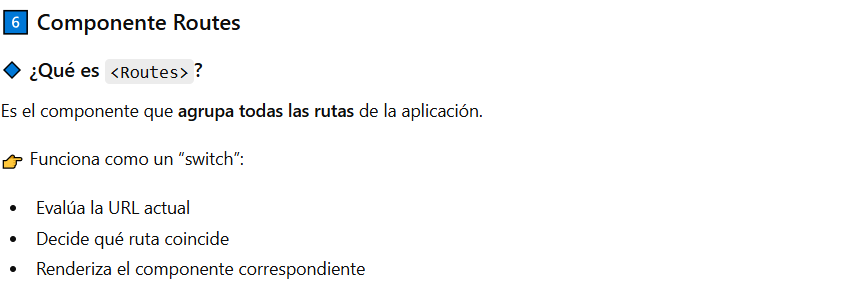


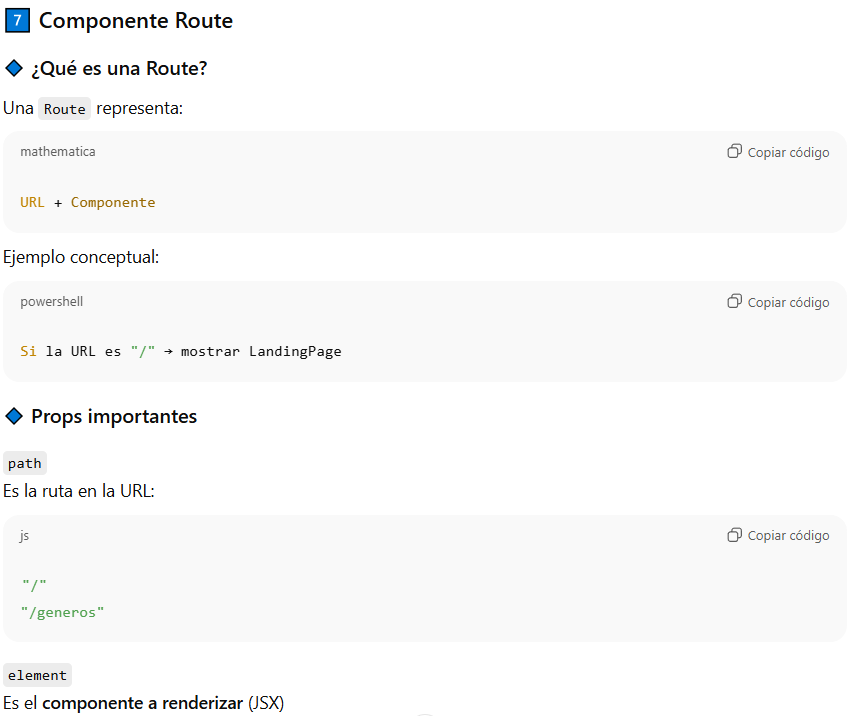




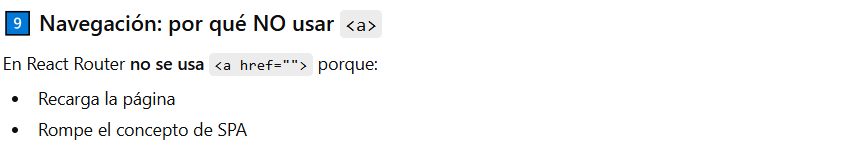


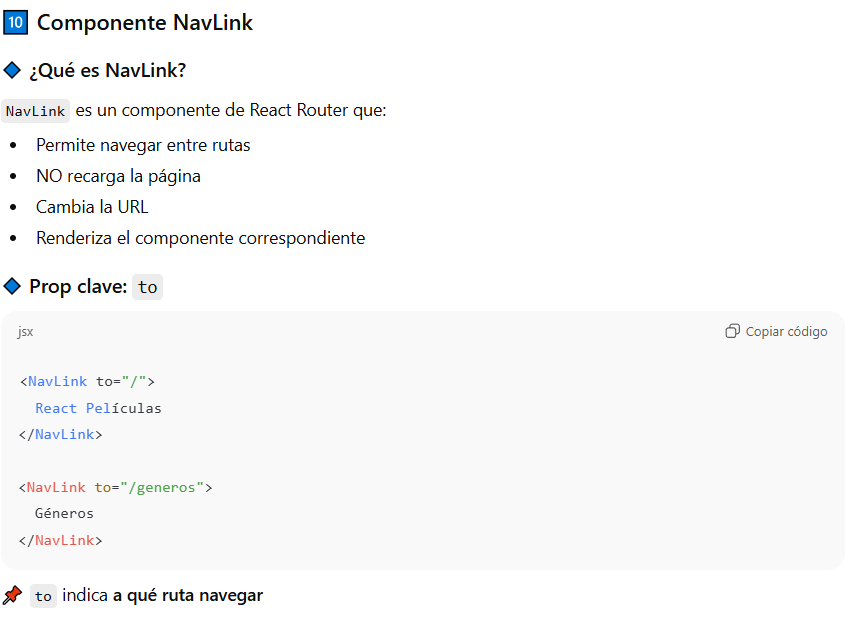


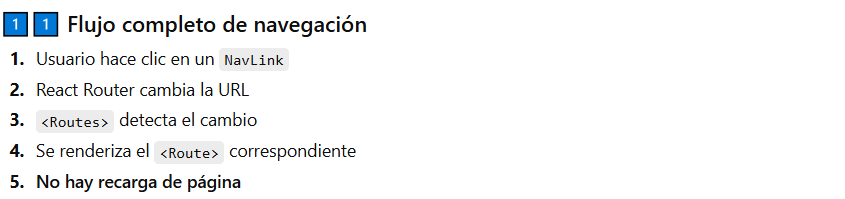


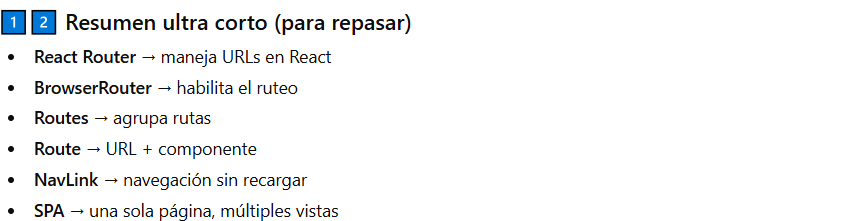










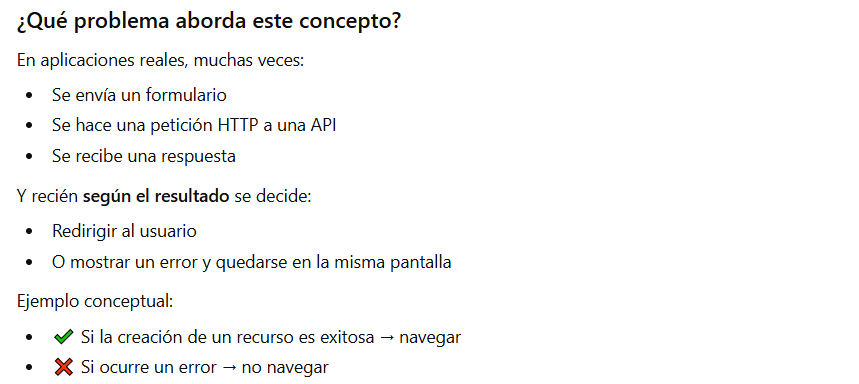


Asi quedaría NavLink con sus respectivos enlaces de redireccionamiento:

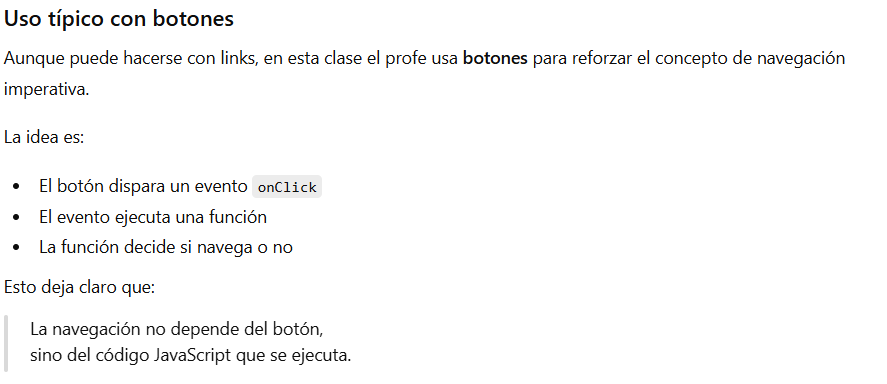


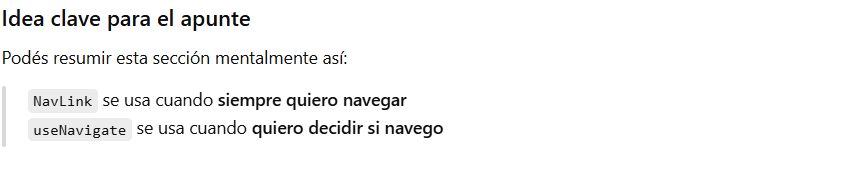
**Navegando con JavaScript**

****

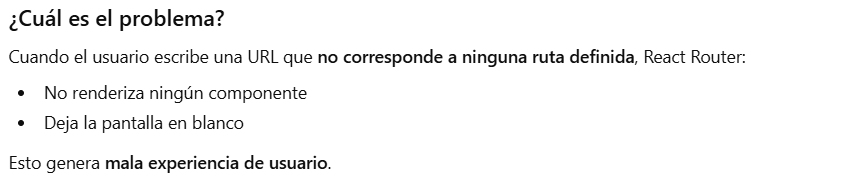
****

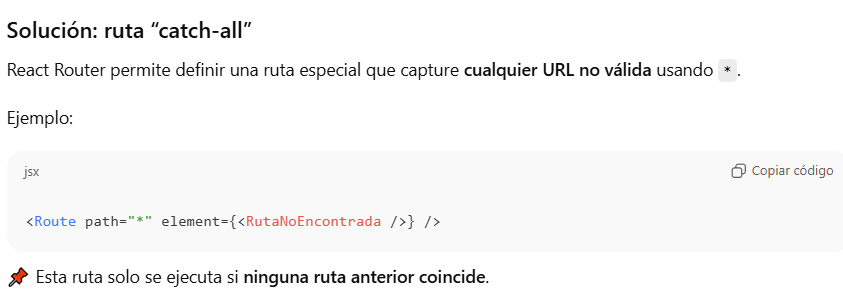
****

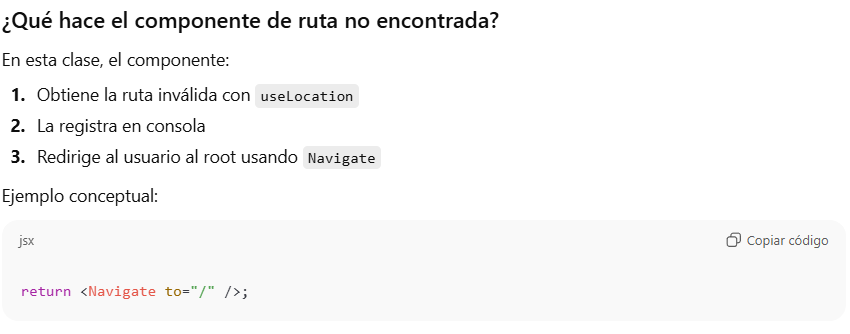
****

****

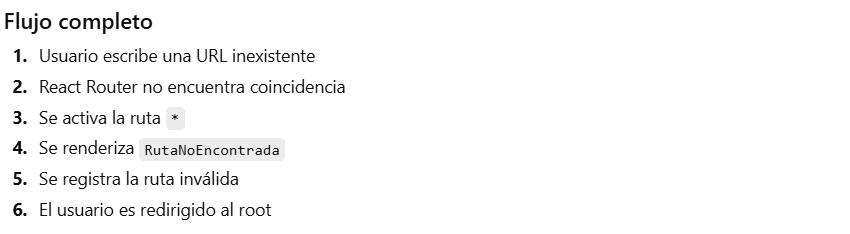
**Manejando Rutas No Encontradas (404**)

****

****

****

****

****