

React.

Cotizador de seguros del coche.

1. Qué es context.

Disponible desde React 16.3, permite pasar el state o funciones desde el componente principal hasta los hijos, sin necesidad de pasarlo por cada componente.

Cuenta con un hook denominado just Context.

Permite actualizar el state desde el hijo, o ejecutar una función que lo actualice.

2. Creando el Formulario.

```
jsersan@iMac-de-Jose seguros % npm create vite@latest
✓ Project name: ... cotizador
✓ Select a framework: > react
✓ Select a variant: > react

Scaffolding project in /Applications/MAMP/htdocs/react/seguros/cotizador...

Done. Now run:

  cd cotizador
  npm install
  npm run dev

jsersan@iMac-de-Jose seguros % █

[jsersan@iMac-de-Jose cotizador % npm i tailwindcss postcss autoprefixer
added 69 packages, and audited 151 packages in 2s

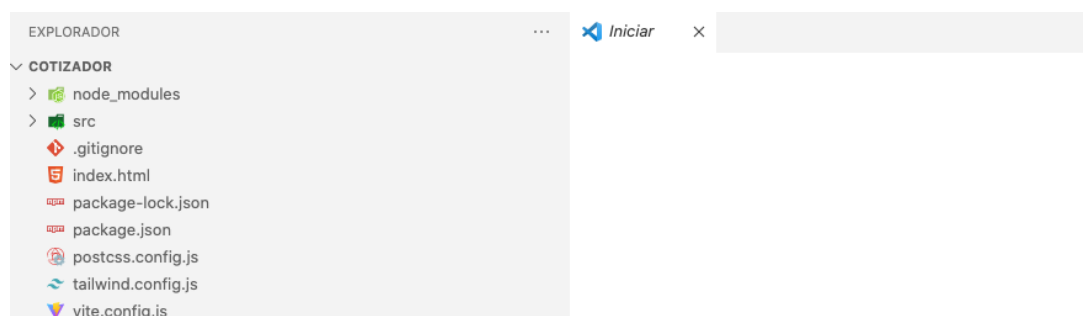
21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
jsersan@iMac-de-Jose cotizador % █
```

Instalamos tailwindcss:

```
jsersan@iMac-de-Jose cotizador % npx tailwindcss init -p
Created Tailwind CSS config file: tailwind.config.js
Created PostCSS config file: postcss.config.js
jsersan@iMac-de-Jose cotizador % █
```

Lo abrimos con Visual Studio Code:



En tailwind.config.js:

```
tailwind.config.js x
tailwind.config.js > ...
1  module.exports = {
2    content: ["index.html", "./src/**/*.jsx"],
3    theme: {
4      extend: {},
5    },
6    plugins: [],
7  }
```

En index.css:

```
src > index.css
1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;
4
```

Eliminamos logo.svg:

COTIZADOR

node_modules

src

App.css

App.jsx

favicon.svg

index.css

logo.svg

main.jsx

.gitignore

index.html

src > App.jsx > ...

1

2

3 function App() {

4

5 return (

6 | <h1>App Cotizador</h1>

7 |)

8 }

9

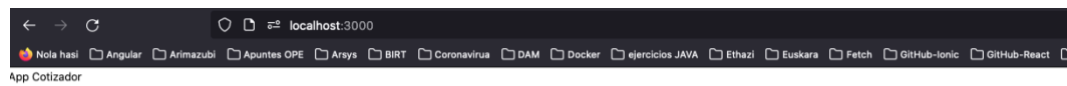
10 export default App

11

Modificamos index.html:

```
index.html > ...
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Cotizador Formularios y Context</title>
8    </head>
9    <body class="bg-gradient-to-r from indigo-500 to indigo-800">
10     <div id="root"></div>
11     <script type="module" src="/src/main.jsx"></script>
12   </body>
13 </html>
```

Lanzamos el proyecto:



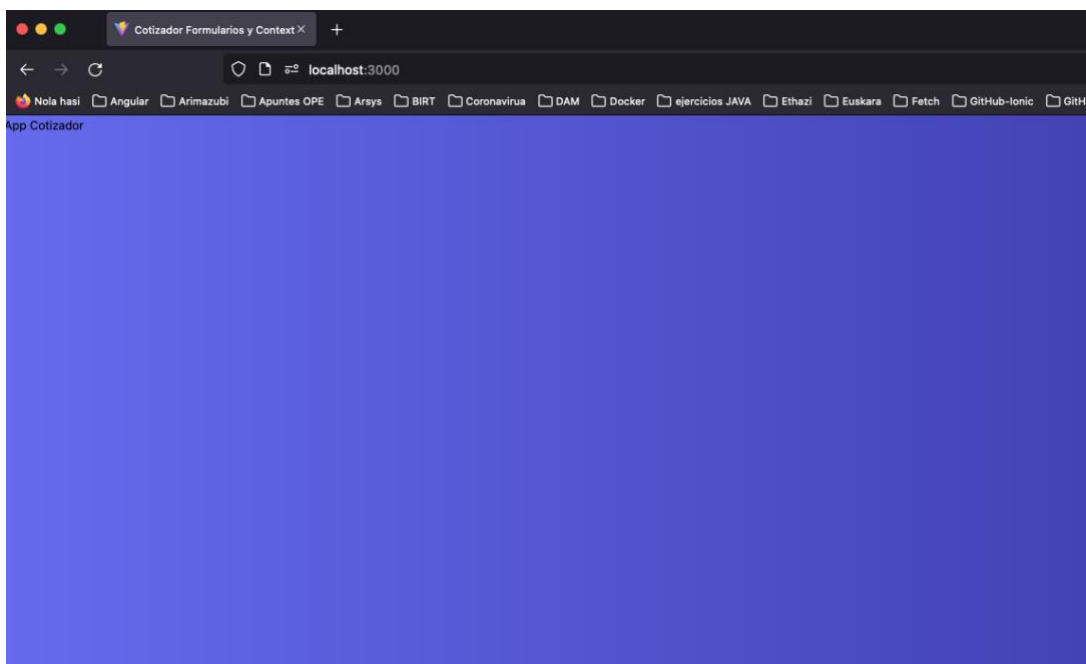
Creamos las carpetas components, constants, context, hooks, styles

```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Cotizador Formularios y Context</title>
8    </head>
9    <body class="bg-gradient-to-r from-indigo-500 to-indigo-800">
10     <div id="root"></div>
11     <script type="module" src="/src/main.jsx"></script>
12   </body>
13 </html>
```

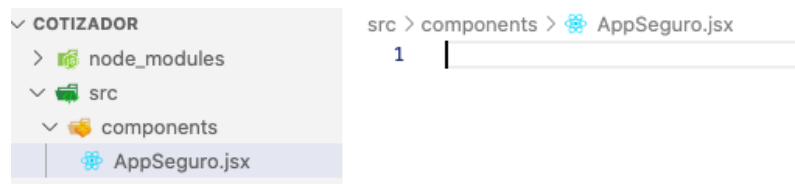
Como no tiene el color deseado vuelvo a revisar la instalación de librerías:

```
jsersan@iMac-de-Jose cotizador % npm i tailwindcss postcss autoprefixer
up to date, audited 151 packages in 2s
21 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
jsersan@iMac-de-Jose cotizador %
```

Resultado: gradiente.



Creamos en components un. Archivo denominado AppSeguro.jsx



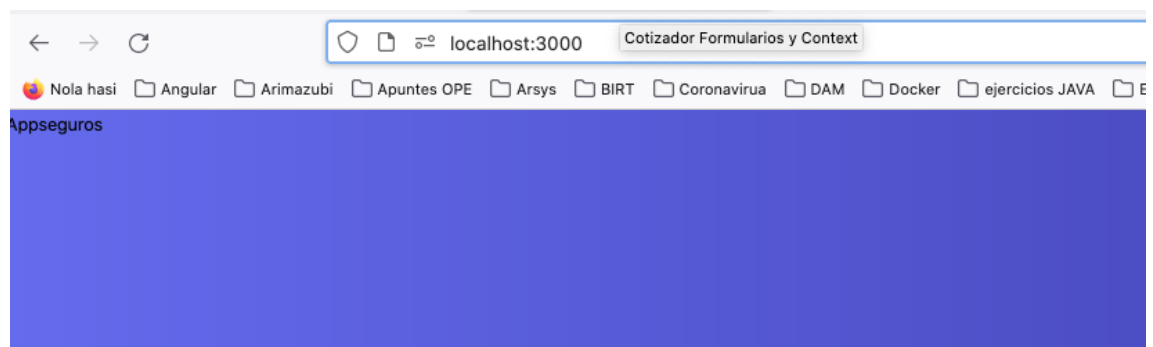
La estructura de este archivo es:

```
src > components > AppSeguro.jsx > default
1  import React from "react";
2
3  const AppSeguro = () => {
4    return (
5      <div>Appseguros</div>
6    )
7  }
8
9  export default AppSeguro
```

Así en App.jsx:

```
tailwind.config.js  index.css  App.jsx  AppSeguro
src > App.jsx > ...
1  import AppSeguro from './components/AppSeguro'
2
3  function App() {
4
5    return (
6      <AppSeguro/>
7    )
8  }
9
10 export default App
11
```

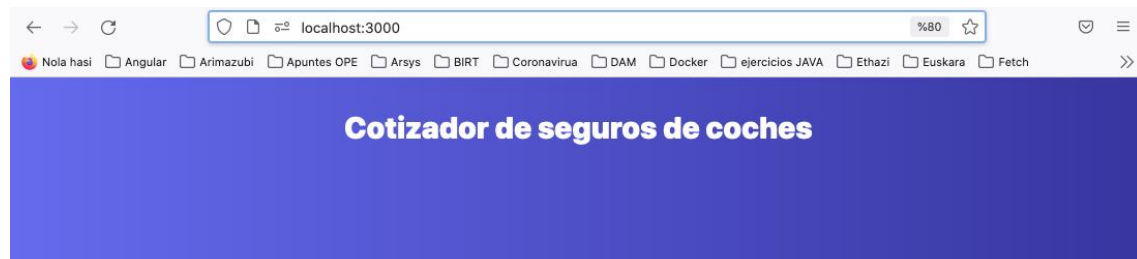
Así en la página:



Así en AppSeguros.jsx:

```
src > components > AppSeguro.jsx > AppSeguro
1  import React from "react";
2
3  const AppSeguro = () => {
4    return (<
5      <header className="my-10">
6        <h1 className="text-white text-center text-4xl font-black">
7          Cotizador de seguros de coches
8        </h1>
9      </header>
10    </header>
```

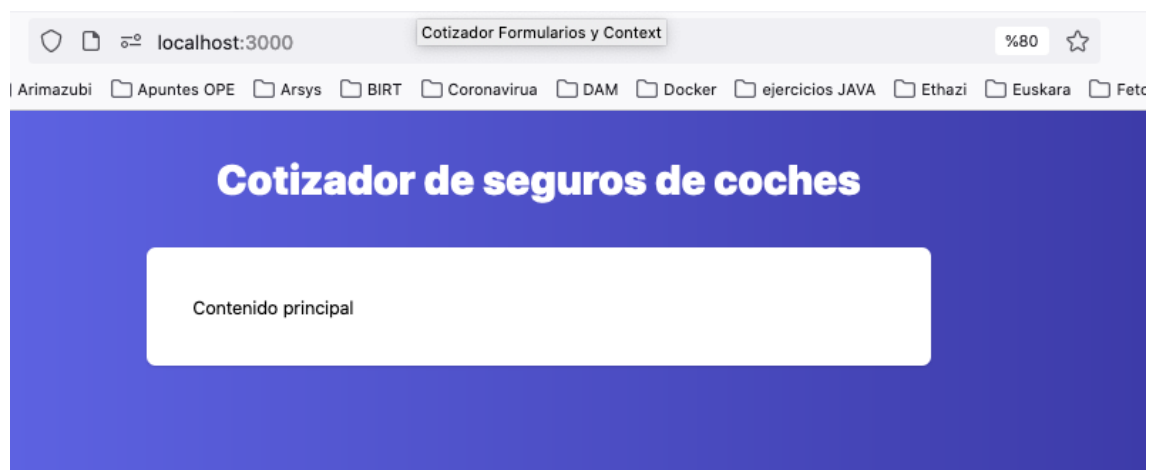
Resultado:



Seguimos:

```
3  const AppSeguro = () => {
4    return (<
5      <header className="my-10">
6        <h1 className="text-white text-center text-4xl font-black">
7          Cotizador de seguros de coches
8        </h1>
9      </header>
10
11      <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
12        <p>Contenido principal</p>
13      </main>
14    </main>
```

Resultado:



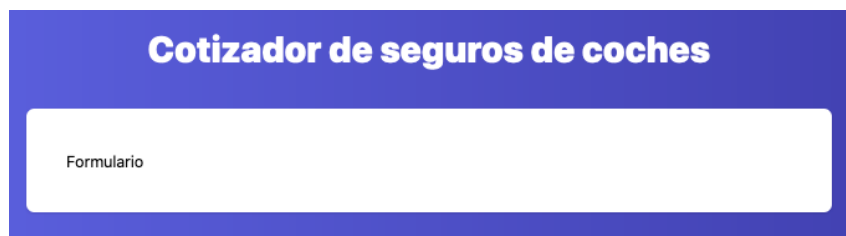
Donde está el contenido principal vamos a tener un formulario:

```
src > components > AppSeguro.jsx > AppSeguro
1  import React from "react";
2  import Formulario from "../Formulario";
3
4  const AppSeguro = () => {
5    return (
6      <header className="my-10">
7        <h1 className="text-white text-center text-4xl font-black">
8          Cotizador de seguros de coches
9        </h1>
10     </header>
11
12     <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
13       <Formulario/>
14     </main>
15   )
16 }
```

Y el componente formulario:

```
src > components > Formulario.jsx > ...
1
2
3  const Formulario = () => {
4    return (
5      <div>Formulario</div>
6    )
7  }
8
9  export default Formulario
```

Vista previa:



Hacemos el formulario:

```
2
3  const Formulario = () => {
4    return (
5      <form>
6        <div className="my-5">
7          <label className="block mb-3 font-bold text-gray-400 uppercase">--
8        </label>
9        <select
10         name="marca"
11         className="w-full p-3 bg-white border border-gray-200">
12          <option value="">--Seleccionar Marca --</option>
13        </select>
14        </div>
15      </form>
16    )
17  }
18  export default Formulario
```

Resultado:

MARCA

--Seleccionar Marca --

Para los datos, nos creamos un archivo constant/index.js:



Este archivo contendrá un arreglo de marcas:

```
src > constants > JS index.js > ...
1  export const MARCAS = [
2    { id:1, nombre: 'Europeo'},
3    { id:2, nombre: 'Americano'},
4    { id:3, nombre: 'Asiático'},
5  ];
```

Lo importamos en Formulario.jsx:

```
3  const Formulario = () => {
4    return (
5      <div>
6        <form>
7          <div className="my-5">
8            <label className="block mb-3 font-bold text-gray-400 uppercase">--
9            </label>
10           <select
11             name="marca"
12             className="w-full p-3 bg-white border border-gray-200">
13             <option value="">--Seleccionar Marca --</option>
14             {MARCAS.map(marca => (
15               <option
16                 key={marca.id}
17                 value={marca.id}
18               >
19                 {marca.nombre}
20               </option>
21             ))}
22           </select>
23         </div>
24       </form>
25     </div>
26   )
27 }
28
29 }
```

Resultado:

Cotizador de seguros de coches

--Seleccionar Marca --

☒ Europeo
 ☐ Americano
 ☐ Asiático

Calculamos cuáles fueron los últimos 20 años:

```
src > constants > JS index.js > ...
1  export const MARCAS = [
2    { id:1, nombre: 'Europeo'},
3    { id:2, nombre: 'Americano'},
4    { id:3, nombre: 'Asiático'},
5  ];
6
7  const YEARMAX = new Date().getFullYear();
8
9  export const YEARS = Array.from( new Array(20), (valor, index) => YEARMAX - index )
```

Esta constante la utilizamos en el formulario Formulario.jsx:

```
src > components > Formulario.jsx > ...
1  import { MARCAS, YEARS } from "../constants"
2
3  const Formulario = () => {
4    return (
5      <>
6      <form>
```

Copiamos el primer div en un segundo y en el select en vez de marcas colocamos YEARS:

```
26  <div className="my-5">
27    <label className="block mb-3 font-bold text-gray-400 uppercase">
28      Año
29    </label>
30    <select
31      name="marca"
32      className="w-full p-3 bg-white border border-gray-200">
33
34      <option value="">--Seleccionar Año --</option>
35      {YEARS.map(year =>{
36        <option
37          key={year}
38          value={year}
39        >
40          {year}
41        </option>
42      })}
43    </select>
44  </div>
```

Resultado:

The screenshot shows a web form with two dropdown menus. The first dropdown, labeled 'MARCA', has a placeholder text '--Seleccionar Marca --'. The second dropdown, labeled 'AÑO', has a placeholder text '✓ --Seleccionar Año --' and is currently open, displaying a list of years: 2022, 2021, 2020, 2019, 2018, 2017, and 2016.

Tendremos un tercer parámetro que serán los planes. Así, en index.js:

```
--
14 export const PLANES = [
15   { id:1, nombre: 'Básico'},
16   { id:2, nombre: 'Completo'},
17 ];
```

Lo asociamos a un fragment que hay que importarlo:

```
src > components > Formulario.jsx > Formulario > PLANES.map() callback
1 import { Fragment } from 'react'
2 import { MARCAS, YEARS, PLANES } from '../constants'
3
```

Resultado:

```
46 <div className="my-5">
47   <label className="block mb-3 font-bold text-gray-400 uppercase">
48     Elige un plan
49   </label>
50   <div className="flex gap-3 items-center">
51     {PLANES.map(plan =>{
52       <Fragment key={plan.id}>
53         <label>
54           { plan.nombre }
55         </label>
56         <input
57           type="radio"
58           name="plan"
59           value={plan.id}
60         />
61       </Fragment>
62     )}}
63   </div>
64 </div>
65 <input
66   type="submit"
67   className="w-full bg-indigo-500 hover:bg-indigo-600 transition-colors
68     text-white cursor-pointer p-3 uppercase font-bold"
69   value="Cotizar" />
```

Resultado:

MARCA

--Seleccionar Marca --

AÑO

--Seleccionar Año --

ELIGE UN PLAN

Básico ☐ Completo ☒

COTIZAR

3. Creando el context.

Creemos en la carpeta context un fichero CotizadorProvider.jsx. Un provider indica cuál es la fuente de los datos. Hay que exportarlo al final del archivo:

```
src > context > CotizadorProvider.jsx > CotizadorProvider
1  import { createContext } from 'react'
2
3  const CotizadorContext = createContext()
4
5  const CotizadorProvider = ({ children }) => {
6
7      return(
8
9          <CotizadorContext.Provider>
10             {children}
11          </CotizadorContext.Provider>
12      )
13  }
14
15
16  export {
17      CotizadorProvider
18  }
19
20  export default CotizadorContext;
```

Este provider vamos a utilizarlo en App.jsx. Puede verse que el formato del return es el mismo:

```
src > App.jsx > App
1  import { CotizadorProvider } from './context/CotizadorProvider'
2  import AppSeguro from './components/AppSeguro'
3
4  function App() {
5
6      return (
7          <CotizadorProvider>
8              <AppSeguro/>
9          </CotizadorProvider>
10      )
11  }
12
13  export default App
14
```

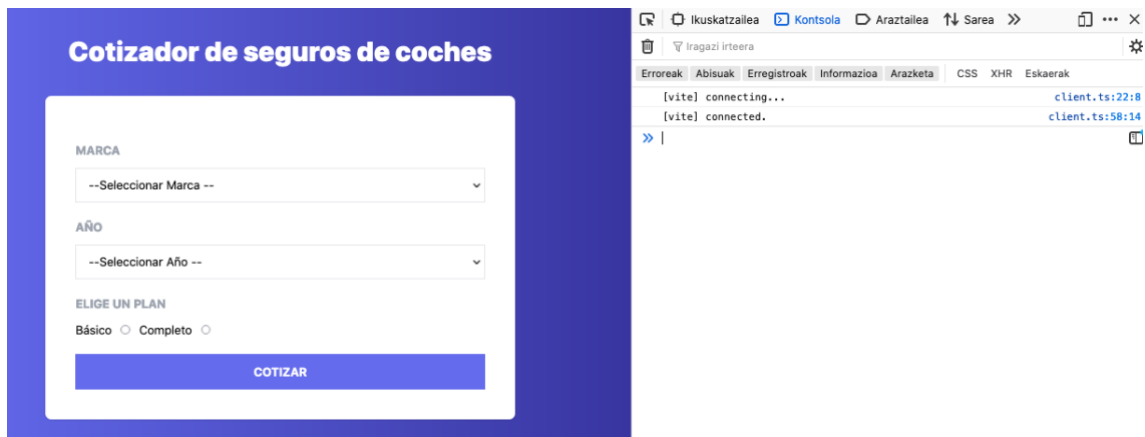
Los datos del provider van a estar accesibles a AppSeguro ya que lo rodea. Guardamos cambios ya aparecerá un error en la consola:

```
Warning: The `value` prop is required for the `<Context.Provider>`. Did you misspell it or forget to pass it?
CotizadorProvider@http://localhost:3000/src/context/CotizadorProvider.jsx?t=1648486219892:19:27
App
```

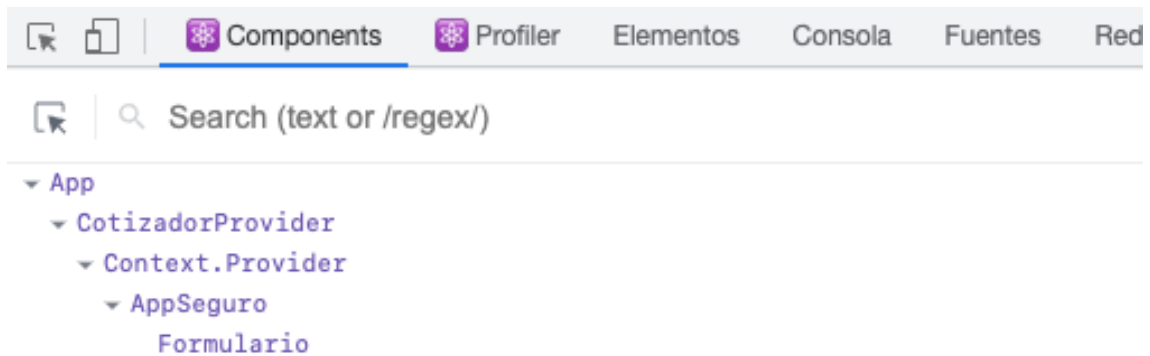
El error se quita asociando un valor al CotizadorContext.Provider que debe devolver:

```
9  <CotizadorContext.Provider
10     value={{ }}
11 >
```

Resultado:



Si utilizamos las herramientas de desarrollo para React que tiene Google Chrome:



Recuerden que para acceder a los elementos del formulario y el cotizadorContext debimos importar useContext para centralizar todo el trabajo del componente. Todo lo que se ponga dentro del value quedará a disposición del resto de componentes.

```

5  const CotizadorProvider = ({ children }) => {
6
7    return(
8      <CotizadorContext.Provider
9        value={{ }}
10      >
11        {children}
12      </CotizadorContext.Provider>
13    )
14  }
15

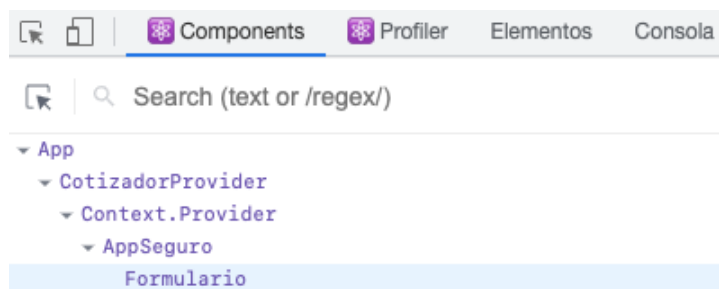
```

4. Paso de datos o funciones en el Provider.

Probamos el paso de datos en el proveedor. Si hacemos:

```
5 const CotizadorProvider = ({ children }) => {
6
7   const hola = "Hola mundo"
8
9   return(
10    <CotizadorContext.Provider
11      value={{ }}
12    >
13      {children}
14    </CotizadorContext.Provider>
15    </CotizadorProvider>
16  )
17 }
```

No estará disponible en ningún lugar sólo en el espacio antes del return.



Para acceder a esta información en components/Formulario.jsx:

```
src > components > Formulario.jsx > Formulario
1 import { Fragment, useContext } from 'react'
2 import { MARCAS, YEARS, PLANES } from '../constants'
3
4 const Formulario = () => {
5
6   const {} = useContext()
7 }
```

Hay que decirle que Context es el que queremos pasar.

En CotizadorProvider.jsx creábamos CotizadorContext:

```
src > context > CotizadorProvider.jsx > CotizadorProvider
1 import { createContext } from 'react'
2
3 const CotizadorContext = createContext()
```

Este CotizadorContext lo importamos en Formulario.jsx

```
src > components > Formulario.jsx > Formulario
1 import { Fragment, useContext } from 'react'
2 import { MARCAS, YEARS, PLANES } from '../constants'
3 import CotizadorContext from '../context/CotizadorProvider'
4
5 const Formulario = () => {
6
7   const {} = useContext(CotizadorContext)
8 }
```

De esta forma va a buscar todo lo que se encuentre dentro de CotizadorContext (en CotizadorProvider):

```

10   <CotizadorContext.Provider
11     value={{ }}
12   >
13     {children}
14   </CotizadorContext.Provider>

```

Para probar la correcta lectura de la constante hola, la extraemos en Formulario.jsx:

```

5   const Formulario = () => {
6
7     const { hola } = useContext(CotizadorContext)
8
9     console.log(hola);
10
11     return (
12       <form>
13

```

Probamos y vemos que nos da undefined:

[vite] hot updated: /src/components/Formulario.jsx	client.ts:352
[vite] hot updated: /src/index.css	client.ts:352
undefined	Formulario.jsx:9

Eso es debido a que el useContext sólo tiene acceso a lo que está dentro del return(). Hacemos así en el CotizadorProvider():

```

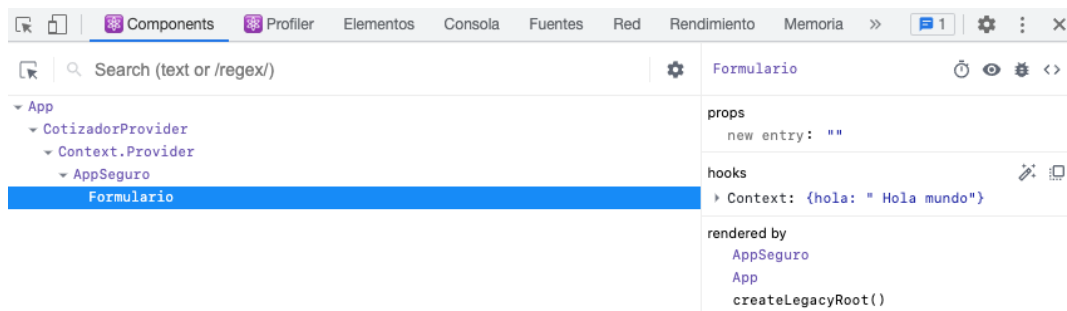
5   const CotizadorProvider = ({ children }) => {
6
7     const hola = "Hola mundo"
8
9     return(
10      <CotizadorContext.Provider
11        value={{
12          hola
13        }}
14      >
15        {children}
16      </CotizadorContext.Provider>
17    )
18  }

```

Resultado:

[vite] connecting...	client.ts:22
[vite] connected.	client.ts:58
Hola mundo	Formulario.jsx:9

Con las herramientas de React del Navegador:



Podemos definir incluso funciones en CotizadorProvider:

```

5  const CotizadorProvider = ({ children }) => {
6
7      const hola =" Hola mundo"
8
9      const fnHolaMundo = () =>{
10         console.log('Hola mundo desde una función')
11     }
12
13     return(
14         <CotizadorContext.Provider
15             value={
16                 {
17                     hola,
18                     fnHolaMundo
19                 }
20             }
21         >{children}</CotizadorContext.Provider>
22     )
23 }

```

Entonces en Formulario:

```

4
5  const Formulario = () => {
6
7      const { hola, fnHolaMundo } = useContext(CotizadorContext)
8
9      console.log(hola);
10
11      fnHolaMundo()
12
13      return (

```

Resultado:



5. Pasar state desde el Provider y Consumirlo en los Componentes.

Vamos a modificar nuestro anterior provider para que suministre datos a nuestra app. De la misma forma que podemos acceder a datos y funciones, podemos compartir states. Así. En context/CotizadorProvider.jsx:

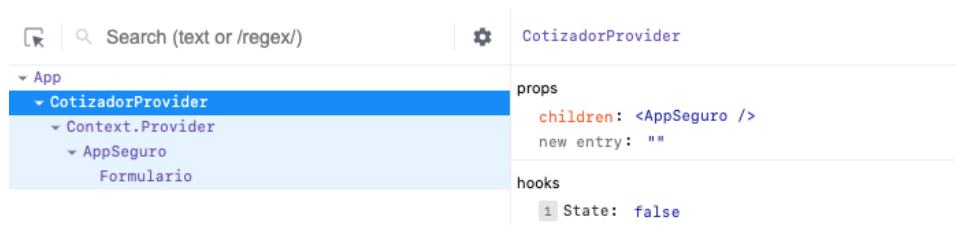
```
3  const CotizadorContext = createContext()
4
5  const CotizadorProvider = ({ children }) => {
6
7    const [modal, setModal] = useState(false);
8  }
```

Y en components/Formulario.jsx:

```
3  import CotizadorContext from '../context/CotizadorProvider'
4
5  const Formulario = () => {
6
7    const { } = useContext(CotizadorContext)
```

Nuestra app:

En la consola podemos ver como está state a false:



En CotizadorProvider.jsx pasamos el modal, setModal al context a través de return:

```

7      const [modal, setModal]= useState(false);
8
9      return(
10         <CotizadorContext.Provider
11           value={{
12             modal,
13             setModal
14           }}
15         >

```

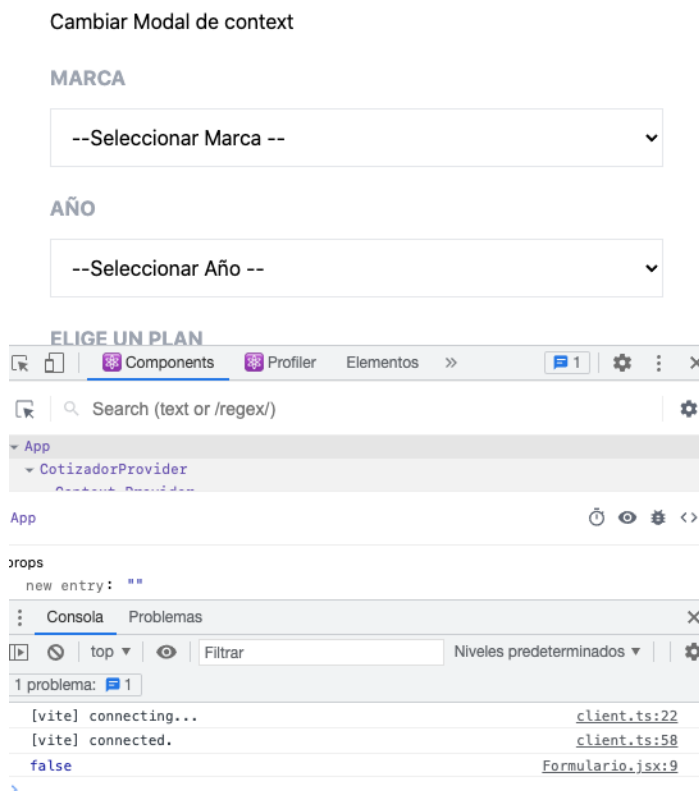
También la extraeremos en Formulario.jsx uso el modal y setModal de esta forma; quitamos el form para que no recargue la página:

```

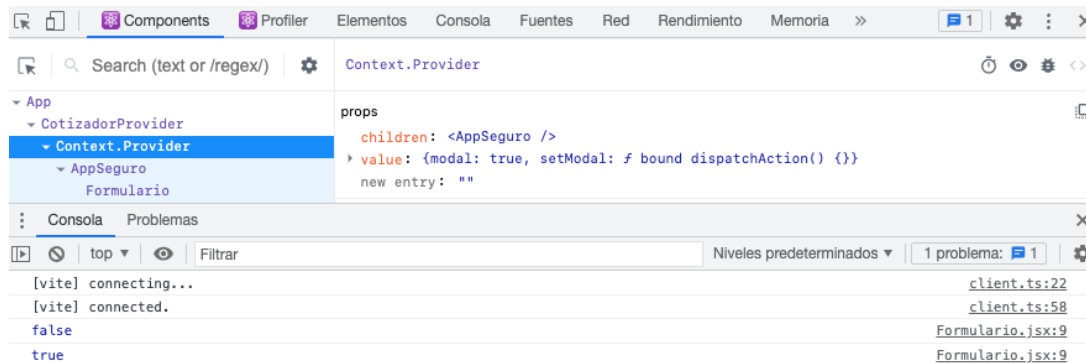
12     return (
13       <div>
14
15         <button
16           onClick={()=> {
17             return setModal(true)
18           }}>
19           Cambiar Modal de context
20         </button>
21       </div>
22     )

```

En la consola tenemos:



Si le damos al botón CambiarModal de Context se cambia a true y recarga la página volviéndose a false.



Podemos hacerlo así, podemos tener una función que tenga el state dentro del provider y así estará disponible para todos los componentes.

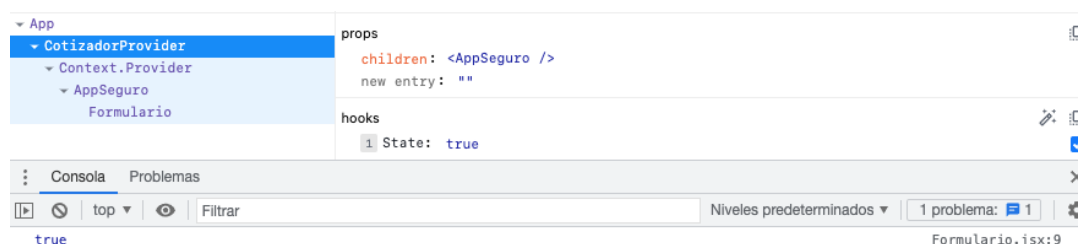
Otra forma es utilizar funciones para cambiar el state como por ejemplo en CotizadorProvider.jsx:

```
5 const CotizadorProvider = ({ children }) => {
6
7   const [modal, setModal] = useState(false);
8
9   const cambiarState = () =>{
10     setModal(!modal)
11   }
12
13   return(
14     <CotizadorContext.Provider
15       value={
16         modal,
17         cambiarState
18       }
19     >
20       {children}
21     </CotizadorContext.Provider>
22   )
23 }
```

La llamamos desde Formulario:

```
7 const {modal, cambiarState} = useContext(CotizadorContext)
8
9 console.log(modal);
10
11
12 return (
13   <div>
14     <button
15       onClick={cambiarState}
16     />
17   </div>
18 )
```

Probamos:



Personalmente me gusta más esta forma. El formulario Formulario.jsx:

```

3  import CotizadorContext from '../context/CotizadorProvider'
4
5  const Formulario = () => {
6
7      const {modal, cambiarState} = useContext(CotizadorContext)
8
9      console.log(modal);
10
11
12     return (
13         <>
14             <button
15                 onClick={cambiarState}
16             >
17                 Cambiar Modal de context
18             </button>
19         </>
20     )
21 }

```

El provider CotizadorProvider.jsx

src > context > CotizadorProvider.jsx > CotizadorProvider > cambiarState

```

1  import { useState, createContext } from 'react'
2
3  const CotizadorContext = createContext()
4
5  const CotizadorProvider = ({ children }) => {
6
7      const [modal, setModal] = useState(false);
8
9      const cambiarState = () => {
10         setModal(!modal)
11     }
12
13     return(
14         <CotizadorContext.Provider
15             value={{
16                 modal,
17                 cambiarState
18             }}
19         >
20             {children}
21         </CotizadorContext.Provider>
22     )
23 }

```

6. Creando un Hook para acceder a los datos del Provider.

Si queremos acceder a un modal, state o funciones, de un provider la mejor forma es el uso de hooks. Vamos a la carpeta hook y creamos un hook:

```

useCotizador.jsx x App.jsx CotizadorProvider.jsx tailwind.c
src > hooks > useCotizador.jsx
1  import { useContext } from 'react';
2
3

```

Lo apropiado es ir separando la lógica de programación y tener un hook para cada uno de los context.

```

useCotizador.jsx x App.jsx CotizadorProvider.jsx tailwind.con
src > hooks > useCotizador.jsx > useCotizador
1  import { useContext } from 'react';
2  import CotizadorContext from '../context/CotizadorProvider';
3
4  const useCotizador = () => {
5    return useContext(CotizadorContext);
6  }
7
8  export default useCotizador
9

```

De esta forma, cuando lo llame, va a buscar en el context, va a montar el context y de esta forma ya podremos acceder a las funciones del formulario modal y cambiarState:

```

5  const Formulario = () => {
6
7    const {modal, cambiarState} = useContext(CotizadorContext)
8

```

Ahora en formulario ya no es necesario utilizar el useContext:

```

src > components > Formulario.jsx > Formulario
1  import { Fragment } from 'react'
2  import { MARCAS, YEARS, PLANES } from '../constants'
3  import useCotizador from '../hooks/useCotizador';
4
5  const Formulario = () => {
6
7    const {modal, cambiarState} = useCotizador()
8
9    console.log(modal);
--

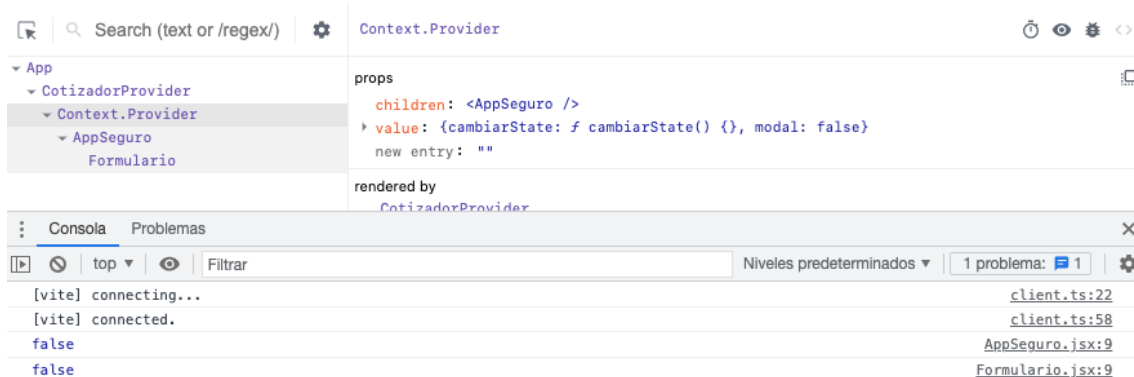
```

En la consola del navegador:

En AppSeguro.jsx ya no es necesario utilizar el useContext:

```
src > components > AppSeguro.jsx > AppSeguro
1
2
3 import Formulario from "../Formulario";
4 import useCotizador from "../hooks/useCotizador";
5
6 const AppSeguro = () => {
7
8   const { modal } = useCotizador({
9     console.log(modal);
10  });
11 }
```

Lanzamos la aplicación y en la consola:



The screenshot shows the React DevTools component inspector and console. The component inspector shows the AppSeguro component rendered by CotizadorProvider. The console shows the output of the application.

Component Inspector:

- App
 - CotizadorProvider
 - Context.Provider
 - AppSeguro
 - Formulario

Props:

- children: <AppSeguro />
- value: {cambiarState: f cambiarState() {}, modal: false}
- new entry: ""

rendered by: CotizadorProvider

Console:

```
[vite] connecting...
[vite] connected.
false
false
```

Ahora en CotizadorProvider.jsx:

```
src > context > CotizadorProvider.jsx > CotizadorProvider
1 import { useState, createContext } from 'react'
2
3 const CotizadorContext = createContext()
4
5 const CotizadorProvider = ({ children }) => {
6
7   return(
8     <CotizadorContext.Provider
9       value={{
10         }}
11     >
12       {children}
13     </CotizadorContext.Provider>
14   )
15 }
16
```

En Formulario.jsx eliminamos lo de useContext:

```
src > components > Formulario.jsx > Formulario
1 import { Fragment } from 'react'
2 import { MARCAS, YEARS, PLANES } from "../constants"
3 import useCotizador from "../hooks/useCotizador";
4
5 const Formulario = () => {
6
```

Ahora en AppSeguro.jsx:

src > components > AppSeguro.jsx > ...

```

1
2
3 import Formulario from "../Formulario";
4
5 const AppSeguro = () => {
6
7   return (
8     <>

```

Nuestra app presenta problemas ahora porque no encuentra cambiarState:

```

> Uncaught ReferenceError: cambiarState is not defined @react-refresh:317
    at Formulario (Formulario.jsx:13:27)
    at renderWithHooks (react-dom.development.js:14985:18)
    at mountIndeterminateComponent (react-dom.development.js:17811:13)
    at beginWork (react-dom.development.js:19049:16)
    at HTMLUnknownElement.callCallback2 (react-dom.development.js:3945:14)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:3994:16)
    at invokeGuardedCallback (react-dom.development.js:4056:31)
    at beginWork$1 (react-dom.development.js:23964:7)
    at performUnitOfWork (react-dom.development.js:22776:12)
    at workLoopSync (react-dom.development.js:22707:5)

```

Eliminamos cambiarState del formulario Formulario.jsx:

```

9
10 return (
11   <>
12
13
14   <div className="my-5">
15     <label className="block mb-3 font-bold text-gray-400 uppercase">...

```

Recargamos:

The screenshot shows the React DevTools interface. The top bar includes tabs for Components, Profiler, Elements, Console, Sources, Red, Performance, and Memory. The Components panel is active, showing a tree structure: App > CotizadorProvider > Context.Provider > AppSeguro > Formulario. The Formulario component is selected, showing its props (new entry: "") and rendered by AppSeguro and App. The Console panel is also visible, showing two messages: '[vite] connecting...' and '[vite] connected.' with file references to client.ts:22 and client.ts:58.

En la siguiente sección asociaremos los states a los combos de marca y año. Así como a los botones de radio que determina qué tipo de plan se quiere contratar.

7. Creando el State para el Formulario.

En CotizadorProvider() definimos un manejador de eventos handleChangeDatos:

```

5  const CotizadorProvider = ({ children }) => {
6
7      const handleChangeDatos = e => {
8          console.log(e.target.name);
9          console.log(e.target.value);
10     }
11
12     return(
13         <CotizadorContext.Provider
14             value={{
15                 handleChangeDatos
16             }}
17         >
    
```

Ahora en el formulario como ya tenemos useCotizador lo utilizamos.

```

src > components > Formulario.jsx > Formulario
1  import { Fragment } from 'react'
2  import { MARCAS, YEARS, PLANES } from '../constants'
3  import useCotizador from '../hooks/useCotizador';
4
5  const Formulario = () => {
6
7      const { handleChangeDatos } = useCotizador()
    
```

Lo asociamos a cada select del formulario:

```

18     <select
19         name="marca"
20         className="w-full p-3 bg-white border border-gray-200"
21         onChange={ e => handleChangeDatos(e) } >
22
23         <option value="">--Seleccionar Marca --</option>
24         {MARCAS.map(marca => (
25             <option
26                 key={marca.id}
27                 value={marca.id}
28             >
29                 {marca.nombre}
30             </option>
31         ))}
32     </select>
33 </div>
34 <div className="my-5">
35     <label className="block mb-3 font-bold text-gray-400 uppercase">
36         Año
37     </label>
38     <select
39         name="year"
40         className="w-full p-3 bg-white border border-gray-200"
41         onChange={ e => handleChangeDatos(e) } >
    
```

De esta forma cuando cambiemos el valor del select, lanzaremos un evento. Lo mismo haremos con el option button.

De esta forma:

```
<input
  type="radio"
  name="plan"
  value={plan.id}
  onChange={e => handleChangeDatos(e)}
/>
```

Como se ve, no es necesario crear una variable para cada elemento sino que tenemos un hook para todos los componentes. En la consola:

Consola	Problemas
marca	CotizadorProvider.jsx:8
1	CotizadorProvider.jsx:9
marca	CotizadorProvider.jsx:8
3	CotizadorProvider.jsx:9

Seleccionamos primero europeo (valor=1) y luego asiático (valor=3), capturamos correctamente el mismo a través del hook. Recordemos que en manejador tenemos:

```
7   const handleChangeDatos = e => {
8     console.log(e.target.name);
9     console.log(e.target.value);
10  }
```

Si ahora seleccionamos el año:

marca	CotizadorProvider.jsx:8
1	CotizadorProvider.jsx:9
marca	CotizadorProvider.jsx:8
3	CotizadorProvider.jsx:9
year	CotizadorProvider.jsx:8
2016	CotizadorProvider.jsx:9

Y con el plan:

year	CotizadorProvider.jsx:8
2016	CotizadorProvider.jsx:9
plan	CotizadorProvider.jsx:8
2	CotizadorProvider.jsx:9
plan	CotizadorProvider.jsx:8
1	CotizadorProvider.jsx:9

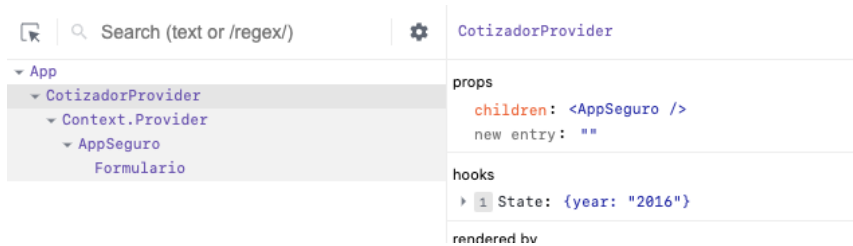
Estos valores debemos definirlos en el provider para hacerlos accesibles al resto de los componentes:

```
5   const CotizadorProvider = ({ children }) => {
6
7     const [ datos, setDatos ] = useState({
8       marca: '',
9       year: '',
10      plan: ''
11    })
```

Así ahora en la función `handleChangeDatos` del provider:

```
13   const handleChangeDatos = e => {
14     setDatos({
15       [e.target.name] : e.target.value
16     })
17   }
18 }
```

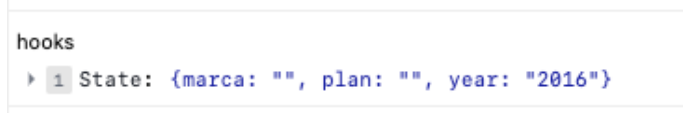
Así en la consola de react si seleccionamos el año 2016 tenemos:



Se ha borrado el contenido de `marca` y `plan`. Para que esto no ocurra en la función `setDatos` debemos agregar el contenido de lo que había antes en el hook:

```
13   const handleChangeDatos = e => {
14     setDatos({
15       ...datos,
16       [e.target.name] : e.target.value
17     })
18   }
19 }
```

El contenido del hook ahora:



Estos datos los vamos a devolver al formulario:

```
20   return(
21     <CotizadorContext.Provider
22       value={{
23         datos,
24         handleChangeDatos
25       }}
26     >
27       {children}
28     </CotizadorContext.Provider>
29   )
```

Volvemos al formulario. Lo importamos a través del provider:

```
3   import useCotizador from '../hooks/useCotizador';
4
5   const Formulario = () => {
6
7     const { datos, handleChangeDatos } = useCotizador()
8   }
```

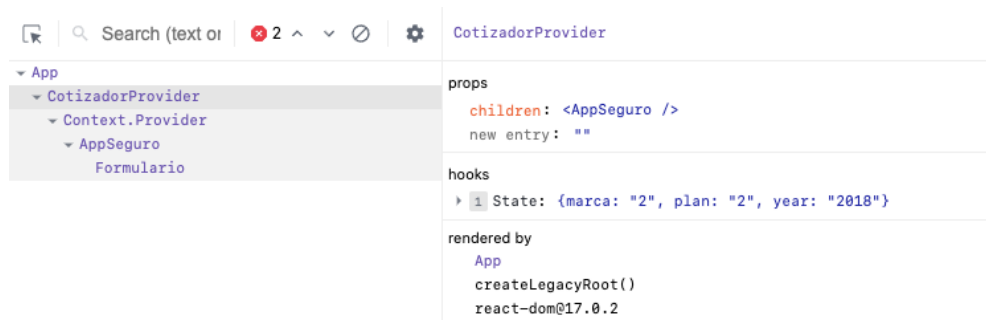

En los select serán `datos.marca` y `datos.year`:

```

14 <div className="my-5">
15   <label className="block mb-3 font-bold text-gray-400 uppercase">--
16   </label>
17   <select
18     name="marca"
19     className="w-full p-3 bg-white border border-gray-200"
20     onChange={ e=> handleChangeDatos(e)}
21     value={datos.marca}>
22
23     <option value="">--Seleccionar Marca --</option>
24     {MARCAS.map(marca =>({
25       <option
26         key={marca.id}
27         value={marca.id}
28       >
29         {marca.nombre}
30       </option>
31     ))}
32   </select>
33 </div>
34 <div className="my-5">
35   <label className="block mb-3 font-bold text-gray-400 uppercase">
36     Año
37   </label>
38   <select
39     name="year"
40     className="w-full p-3 bg-white border border-gray-200"
41     onChange={ e=> handleChangeDatos(e)}
42     value={datos.year}>

```

Para una marca americana, año 2018 y plan completo:



8. Validando Formularios.

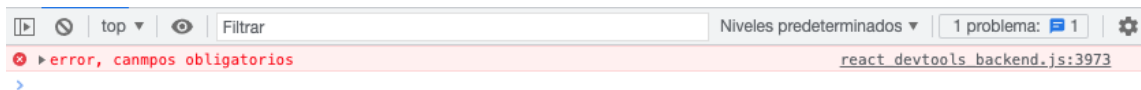
Si queremos validar el formulario, cuando le demos al botón submit entonces la lógica de programación la haremos únicamente en Formulario.jsx

```

5  const Formulario = () => {
6    const { datos, handleChangeDatos } = useCotizador();
7
8    const handleSubmit = (e) => {
9      e.preventDefault();
10
11      if(Object.values(datos).includes('')){ // El campo está vacío))
12        console.error('error, campos obligatorios');
13      }
14
15    };
16  }

```

Probamos:



Este error lo vamos a definir en state del provider:

```

5  const CotizadorProvider = ({ children }) => {
6
7    const [ datos, setDatos ] = useState({
8      marca: '',
9      year: '',
10     plan: ''
11   })
12
13   const [error, setError] = useState('')
14
15   const handleChangeDatos = e => {
16     setDatos({
17       ...datos,
18       [e.target.name] : e.target.value
19     })
20   }
21
22   return(
23     <CotizadorContext.Provider
24       value={{
25         datos,
26         handleChangeDatos,
27         error,
28         setError
29       }}

```

Así, en el formulario:

```

8    const handleSubmit = (e) => {
9      e.preventDefault();
10
11      if(Object.values(datos).includes('')){ // El campo está vacío))
12        setError('Todos los campos son obligatorios');
13      }

```

Debemos importar setError en el formulario:

```

4
5 const Formulario = () => {
6   const { datos, handleChangeDatos, error, setError } = useCotizador();
7
8   const handleSubmit = (e) => {
9     e.preventDefault();
10
11     if(Object.values(datos).includes('')){ // El campo está vacío))
12       setError('Todos los campos son obligatorios')
13       return
14     }
15
16   };
17
18   return (
19     <>
20     { error && }
    
```

Vamos a devolver error y un componente nuevo de Componentts/Error.jsx

```

src > components > Error.jsx > default
1 import useCotizador from "../hooks/useCotizador"
2
3 const Error = () =>{
4   const { error } = useCotizador()
5   return (
6     <div>{error}</div>
7   )
8 }
9
10 export default Error
    
```

Lo utilizamos en el formulario anterior:

```

4 import Error from './Error'
5
6 const Formulario = () => {
7   const { datos, handleChangeDatos, error, setError } = useCotizador();
8
9   const handleSubmit = (e) => {
10     e.preventDefault();
11
12     if(Object.values(datos).includes('')){ // El campo está vacío))
13       setError('Todos los campos son obligatorios')
14       return
15     }
16
17   };
18
19   return (
20     <>
21     { error && <Error/>}
22     <form
23       onSubmit={handleSubmit}
24     >
25     <div className="mv-5">
    
```

Probamos e intentamos mandar el formulario vacío:

Todos los campos son obligatorios

MARCA

--Seleccionar Marca --

AÑO

--Seleccionar Año --

ELIGE UN PLAN

Básico ☐ Completo ☐

COTIZAR

Para que la alerta se vea bien, vamos a darles unos estilos:

```

3  const Error = () =>{
4    const { error } = useCotizador()
5    return (
6      <div className="border text-center border-red-400 bg-red-100 py-3 text-red-700">
7        <p>{error}</p>
8      </div>
9    )
10 }

```

Apariencia:

Todos los campos son obligatorios

MARCA

--Seleccionar Marca --

Este div debe desaparecer cuando estén rellenos todos los campos:

```

9  const handleSubmit = (e) => {
10    e.preventDefault();
11
12    if(Object.values(datos).includes('')){ // El campo está vacío))
13      setError('Todos los campos son obligatorios')
14      return
15    }
16    setError('');

```

Ahora ya se elimina el mensaje de error si el formulario está lleno.

9. Creando la Función para Cotizar el Seguro.

En el provider creamos esta función:

```

22     const cotizarSeguro = () =>{
23       console.log('Cotizando....')
24     }
25
26     return(
27       <CotizadorContext.Provider
28         value={{
29           datos,
30           handleChangeDatos,
31           error,
32           setError,
33           cotizarSeguro
34         }}
35     >

```

La importamos en el formulario:

```

6   const Formulario = () => {
7     const { datos, handleChangeDatos, error, setError, cotizarSeguro } = useCotizador();
8
9     const handleSubmit = (e) => {
10      e.preventDefault();
11
12      if(Object.values(datos).includes('')){ // El campo está vacío)
13        setError('Todos los campos son obligatorios')
14        return
15      }
16      setError('');
17
18      cotizarSeguro();
19    };
20  };

```

Vamos a programar el cálculo del precio del seguro:

```

22     const cotizarSeguro = () =>{
23       // Precio base
24
25
26       // Diferencia en años: hay que restar el 3% por cada año
27
28       // Americano 15%
29       // Europeo 30%
30       // Asiático 5%
31
32
33       // Básico 20%
34       // Completo 50%
35     }

```

10. Trabajando con el Algoritmo de Cotización.

Nos creamos un nuevo componente index.js, un helper que pondremos en la carpeta helpers:



Ahora seguimos con la función cotizarSeguro del provider:

```

22  const cotizarSeguro = () => {
23    // Precio base
24
25    let resultado = 200;
26
27
28    // Diferencia en años: hay que restar el 3% por cada año
29    console.log(datos.year);
30  }

```

Probamos:



Para calcular la diferencia con el año actual en el helper index.js:

```

src > helpers > JS index.js > obtenerDiferenciaYear
1  export function obtenerDiferenciaYear(year) {
2    return new Date().getFullYear() - year;
3
4

```

Lo utilizamos en el provider:

```

src > context > CotizadorProvider.jsx > CotizadorProvider
1  import { useState, createContext } from 'react'
2  import { obtenerDiferenciaYear } from '../helpers'
3

```

Así:

```

23   const cotizarSeguro = () => {
24     // Precio base
25
26     let resultado = 200;
27
28
29     // Diferencia en años: hay que restar el 3% por cada año
30
31     const diferencia = obtenerDiferenciaYear(datos.year);
32     console.log(diferencia);
33   }

```

Resultado:

AÑO

2012

ELIGE UN PLAN

Básico ☐ Completo ☒

COTIZAR

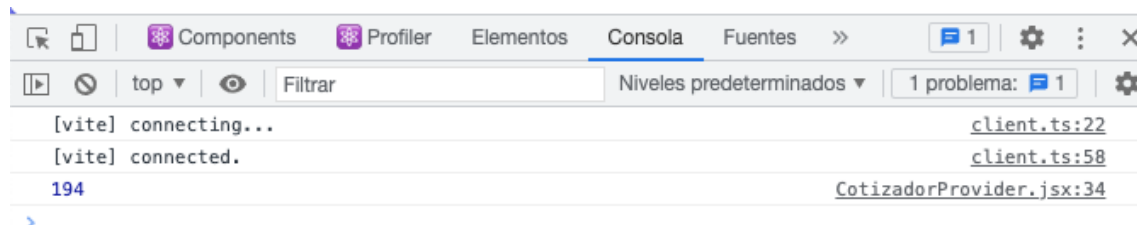
Ahora restamos un 3% por cada año de diferencia:

```

23   const cotizarSeguro = () => {
24     // Precio base
25
26     let resultado = 200;
27
28
29     // Diferencia en años: hay que restar el 3% por cada año
30
31     const diferencia = obtenerDiferenciaYear(datos.year);
32
33     resultado -= resultado*diferencia*0.03
34     console.log(resultado)
35   }

```

Resultado si elegimos americano, año 2021:



Base 200.

Restamos 1*3% de 200 = 6

Resultado: 194

Ahora vamos a ver qué tipo de seguro es: americanos, europeo o asiático. En la carpeta helpers me creo otra función en index.js:

```

5  export function calcularMarca(marca){
6      let incremento;
7
8      switch(marca){
9          case "1":
10             incremento = 1.3
11             break;
12          case "2":
13             incremento = 1.15
14             break;
15          case "3":
16             incremento = 1.05
17             break;
18      }
19      return incremento;
20  }
    
```

La utilizamos en el provider:

```

23  const cotizarSeguro = () =>{
24      // Precio base
25
26      let resultado = 200;
27
28      // Diferencia en años: hay que restar el 3% por cada año
29
30      const diferencia = obtenerDiferenciaYear(datos.year);
31
32      resultado -= resultado*diferencia*0.03
33      console.log(resultado)
34
35      // Americano 15%
36      // Europeo 30%
37      // Asiático 5%
38
39      resultado *= calcularMarca(datos.marca);
40      console.log(resultado)
    
```

Salida:

MARCA

Europeo

▼

AÑO

2022

▼

ELIGE UN PLAN

Básico

☒

Completo

☐

COTIZAR

Components

Profiler

Elementos

Consola

Fuentes

top

▼

Filtrar

Niveles predeterminados

1 problema: 1

[vite] connecting...

[vite] connected.

200

260

client.ts:22

client.ts:58

CotizadorProvider.jsx:33

CotizadorProvider.jsx:40

A ver:

Base: 200

Europeo: $200 * 1,3 = 260$

Y la diferencia en años es $2022 - 2022 = 0$

Resultado = 260 euros.

Si ponemos 2 años:

```

[vite] connecting... client.ts:22
[vite] connected. client.ts:58
188 CotizadorProvider.jsx:33
244.4 CotizadorProvider.jsx:40

```

Base: 200

Diferencia $200 * 0,03 * 2 = 12$

Total: $200 - 12 = 188$ y a esto hay que multiplicarle el coeficiente de la marca que como es europeo es 1,30

Total: $188 * 1,30 = 244,40$ euros

Nos creamos otra función en el helper:

```

22 export function calcularPlan(plan) {
23
24   return plan === "1" ? 1.2 : 1.5;
25
26 }

```

Ahora la usamos en el provider:

```

src > context > CotizadorProvider.jsx > ...
1 import { useState, createContext } from 'react'
2 import { obtenerDiferenciaYear, calcularMarca, calcularPlan } from '../helpers'
3

```

Y en nuestra función cotizarSeguro():

```

42 // Básico 20%
43 // Completo 50%
44
45 resultado *= calcularPlan(datos.plan)
46 console.log(resultado);
47

```

Probamos Europeo, 2022 y completo:

[vite] connecting...	client.ts:22
[vite] connected.	client.ts:58
200	CotizadorProvider.jsx:33
260	CotizadorProvider.jsx:40
390	CotizadorProvider.jsx:46

Si ponemos básico:

200	CotizadorProvider.jsx:33
260	CotizadorProvider.jsx:40
312	CotizadorProvider.jsx:46

Americano, 2010 y Básico:

128	CotizadorProvider.jsx:33
147.2	CotizadorProvider.jsx:40
176.64	CotizadorProvider.jsx:46

En el caso de que queramos tomar dos cifras decimales nada más formateamos la salida en el helper creando esta función:

```

27 export function formatearDinero(cantidad){
28   return cantidad.toLocaleString('es-ES', {
29     style: 'currency',
30     currency: 'EUR'
31   });
32 }

```

La importamos en el provider:

```

src > context > CotizadorProvider.jsx > ...
1 import { useState, createContext } from 'react'
2 import { obtenerDiferenciaYear, calcularMarca, calcularPlan, formatearDinero } from '../helpers'
3

```

Y en la función cotizarSeguro:

```

40
47 // Formatear dinero
48
49 resultado = formatearDinero(resultado);
50 console.log(resultado);
51

```

Salida para Europeo, 2013 y Básico:

[vite] connected.	client.ts:58
146	CotizadorProvider.jsx:33
189.8	CotizadorProvider.jsx:40
227,76 €	CotizadorProvider.jsx:50

Antes de terminar, creamos el siguiente state:

```
13
14     const [error, setError] = useState('')
15     const [resultado, setResultado] = useState(0)
16
```

Al final tenemos:

```
24     const cotizarSeguro = () =>{
25         // Precio base
26
27         let resultado = 200;
28
29         // Diferencia en años: hay que restar el 3% por cada año
30
31         const diferencia = obtenerDiferenciaYear(datos.year);
32
33         resultado -= resultado*diferencia*0.03
34         console.log(resultado)
35
36         // Americano 15%
37         // Europeo 30%
38         // Asiático 5%
39
40         resultado *= calcularMarca(datos.marca);
41         console.log(resultado)
42
43         // Básico 20%
44         // Completo 50%
45
46         resultado *= calcularPlan(datos.plan)
47
48         // Formatear dinero
49
50         resultado = formatearDinero(resultado);
51         setResultado(resultado)
52
53     }
54
```

11. Mostrando el Total de la cotización:

En AppSeguro.jsx importamos este Cotizador:

```

2  import useCotizador from "../hooks/useCotizador";
3  import Formulario from "../Formulario";
4
5  const AppSeguro = () => {
6
7      const { resultado } = useCotizador()
8
9      return (
10         <
11             <header className="my-10">
12                 <h1 className="text-white text-center text-4xl font-black">
13                     Cotizador de seguros de coches
14                 </h1>
15             </header>
16
17             <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
18                 <Formulario/>
19
20                 { resultado}

```

Si probamos todavía no funciona porque, aunque hayamos creado la variable resultado a través del Cotizador, todavía no la hemos pasado. Así en el provider:

```

56  return(
57      <CotizadorContext.Provider
58          value={{
59              datos,
60              handleChangeDatos,
61              error,
62              setError,
63              cotizarSeguro,
64              resultado
65          }}
66      >
67          {children}
68      </CotizadorContext.Provider>

```

Probamos:

MARCA

Americano

AÑO

2021

ELIGE UN PLAN

Básico

☐

Completo

☒

COTIZAR

334,65 €

12. Añadiendo un Spinner de Carga.

Lo que queremos que antes de que cargue el precio final aparezca como una imagen simulando carga. En movimiento:



Tiene la siguiente clase css:

```
.spinner {
  width: 40px;
  height: 40px;
  background-color: #333;

  margin: 100px auto;
  -webkit-animation: sk-rotateplane 1.2s infinite ease-in-out;
  animation: sk-rotateplane 1.2s infinite ease-in-out;
}

@-webkit-keyframes sk-rotateplane {
  0% { -webkit-transform: perspective(120px) }
  50% { -webkit-transform: perspective(120px) rotateY(180deg) }
  100% { -webkit-transform: perspective(120px) rotateY(180deg) rotateX(180deg) }
}

@keyframes sk-rotateplane {
  0% {
    transform: perspective(120px) rotateX(0deg) rotateY(0deg);
    -webkit-transform: perspective(120px) rotateX(0deg) rotateY(0deg)
  } 50% {
    transform: perspective(120px) rotateX(-180.1deg) rotateY(0deg);
    -webkit-transform: perspective(120px) rotateX(-180.1deg) rotateY(0deg)
  } 100% {
    transform: perspective(120px) rotateX(-180deg) rotateY(-179.9deg);
    -webkit-transform: perspective(120px) rotateX(-180deg) rotateY(-179.9deg);
  }
}
```

El tag de uso es:

```
<div class="spinner"></div>
```

Creamos nuevo state (cargando) en CotizadorProvider:

```
14   const [error, setError] = useState('')
15   const [resultado, setResultado] = useState(0)
16   const [cargando, setCargando]=useState(false)
17
```

Lo utilizamos antes de mostrar los resultados. Va a mostrarse durante 3 segundos:

```
53   setCargando(true);
54
55   setTimeout(()=>{
56     setResultado(resultado)
57     setCargando(false)
58   }, 3000)
```

Lo retornamos también:

```

64     return(
65       <CotizadorContext.Provider
66         value={{
67           datos,
68           handleChangeDatos,
69           error,
70           setError,
71           cotizarSeguro,
72           resultado,
73           cargando
74         }}

```

Lo ponemos en AppSeguro.jsx:

```

5  const AppSeguro = () => {
6
7    const { resultado, cargando } = useCotizador()
8
9    return (
10     <>
11       <header className="my-10">
12         <h1 className="text-white text-center text-4xl font-black">
13           Cotizador de seguros de coches
14         </h1>
15       </header>
16
17       <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
18         <Formulario/>
19         { cargando ? 'Cargando...' : resultado }
20       </main>
21     </>
22   )
23
24   </>
25 }

```

Probamos:

MARCA

Americano

AÑO

2018

ELIGE UN PLAN

Básico

☒

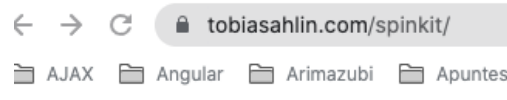
Completo

☐

COTIZAR

Cargando...

Cambiamos el mensaje (Cargando) por una imagen. Creamos el componente Spinner.jsx usando los de esta página:



Usaremos éste:



De la página copio el código del html en Spinner.jsx:

```
src > components > Spinner.jsx > ...
1  import React from "react";
2
3  export const Spinner = () => {
4    return (
5      <div class="spinner">
6        <div class="rect1"></div>
7        <div class="rect2"></div>
8        <div class="rect3"></div>
9        <div class="rect4"></div>
10       <div class="rect5"></div>
11     </div>
12   );
13 }
```

El código css lo copiamos en una styles/Spinner.css:

```
Spinner.css x JS index.js .../helpers Error.jsx AppSeguro.jsx Spinner.
src > styles > Spinner.css > @keyframes sk-stretchdelay
1  .spinner {
2    margin: 100px auto;
3    width: 50px;
4    height: 40px;
5    text-align: center;
6    font-size: 10px;
7  }
8
9  .spinner > div {
10   background-color: #333;
11   height: 100%;
12   width: 6px;
13   display: inline-block;
14
15   -webkit-animation: sk-stretchdelay 1.2s infinite ease-in-out;
16   animation: sk-stretchdelay 1.2s infinite ease-in-out;
17 }
18
19 .spinner .rect2 {
20   -webkit-animation-delay: -1.1s;
21   animation-delay: -1.1s;
```

Lo importamos en Spinner.jsx:

```
src > components > Spinner.jsx > ...
1  import React from "react";
2  import '../styles/Spinner.css'
3
4  export const Spinner = () => {
5    return (
6      <div class="spinner">
7        <div class="rect1"></div>
8        <div class="rect2"></div>
9        <div class="rect3"></div>
10       <div class="rect4"></div>
11       <div class="rect5"></div>
12     </div>
13   );
14 }
```

Importamos este Spinner en AppSeguro.jsx:

```
3  import { Spinner } from "../Spinner";
4  import Formulario from "../Formulario";
5
6  const AppSeguro = () => {
7
8    const { resultado, cargando } = useCotizador()
9
10   return (
11     <>
12       <header className="my-10">
13         <h1 className="text-white text-center text-4xl font-black">
14           Cotizador de seguros de coches
15         </h1>
16       </header>
17
18       <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
19         <Formulario/>
20         { cargando ? <Spinner/>: resultado }
21       </main>
22     </>
23   );
24 }
```

Probamos:




Tras los tres segundos:



Vamos a colocar el resultado en otro sitio. Creamos un componente Resultado. El archivo Resultado.jsx:

```
src > components > Resultado.jsx > Resultado
1  import useCotizador from "../hooks/useCotizador"
2
3  export const Resultado = () => {
4    const { resultado } = useCotizador();
5
6
7    return (
8      <div>{resultado}</div>
9    )
10 }
```

En AppSeguro.jsx:

```
3  import { Spinner } from "../Spinner";
4  import Formulario from "../Formulario";
5  import { Resultado } from "../Resultado";
6
7  const AppSeguro = () => {
8
9    const { resultado, cargando } = useCotizador()
10
11    return (
12      <>
13        <header className="my-10">
14          <h1 className="text-white text-center text-4xl font-black">
15            Cotizador de seguros de coches
16          </h1>
17        </header>
18
19        <main className="bg-white md:w-2/3 lg:w-2/4 mx-auto shadow rounded-lg p-10">
20          <Formulario/>
21          { cargando ? <Spinner/> : <Resultado/> }
22        </main>
23      </>
24    )
25  }
```

Vista previa:

AÑO

--Seleccionar Año --

ELIGE UN PLAN

Básico ☐ Completo ☐

COTIZAR

0

Para que no salga este cero en Resultado.jsx:

```
3  export const Resultado = () => {
4    const { resultado } = useCotizador();
5
6    if ( resultado === 0 ) return null
7  }
```

Ahora está la pantalla limpia de datos espurios:

MARCA

--Seleccionar Marca --

AÑO

--Seleccionar Año --

ELIGE UN PLAN

Básico ☐ Completo ☐

COTIZAR

Probamos Europeo, 2011, completo. Primero simula la carga:

ELIGE UN PLAN

Básico ☒ Completo ☐

COTIZAR



Resultado tras tres segundos de spinner:

MARCA

Europeo

AÑO

2011

ELIGE UN PLAN

Básico ☒ Completo ☐

COTIZAR

209,04 €

13. Añadiendo más Información al Resultado.

En esta sección modificaremos el componente Resultado.jsx para añadir más información:

```

3  export const Resultado = () => {
4      const { resultado, datos } = useCotizador();
5      const { marca, plan, year } = datos;
6
7
8      if ( resultado === 0 ) return null
9
10     return (
11         <div className="bg-gray-100 text-center mt-5 p-5 shadow">
12             <h2 className="text-gray-700 font-black text-3xl">
13                 Resumen
14             </h2>
15         </div>
16     )
17 }

```

Tenemos por ahora:

MARCA

Europeo

AÑO

2021

ELIGE UN PLAN

Básico ☐ Completo ☒

COTIZAR

Resumen

Completamos la información a mostrar con la marca seleccionada:

```

10   return (
11     <div className="bg-gray-100 text-center mt-5 p-5 shadow">
12       <h2 className="text-gray-700 font-black text-3xl">
13         Resumen
14       </h2>
15       <p>
16         <span className="font-bold">Marca:</span>{marca}
17       </p>
18     </div>

```

Si elijo Europeo aparece:



Si queremos importar el nombre de la marca no su valor, en Resultados.jsx importamos MARCAS de constants:

```

src > components > Resultado.jsx > Resultado
1   import useCotizador from "../hooks/useCotizador"
2   import {MARCAS, PLANES } from '../constants/index'
3
4   export const Resultado = () => {
5     const { resultado, datos } = useCotizador();
6     const { marca, plan, year } = datos;
7
8     console.log(MARCAS);
9

```

Probamos:

```

▼ (3) [{...}, {...}, {...}] ⓘ
  ▶ 0: {id: 1, nombre: 'Europeo'}
  ▶ 1: {id: 2, nombre: 'Americano'}
  ▶ 2: {id: 3, nombre: 'Asiático'}
  length: 3
  ▶ [[Prototype]]: Array(0)

```

Podemos ver que los índices son 0,1,2 respectivamente y los ids 1,2 y3. Para solucionar este desfase y no tener que sumar 1 al índice:

```

4   export const Resultado = () => {
5     const { resultado, datos } = useCotizador();
6     const { marca, plan, year } = datos;
7
8     const nombreMarca = MARCAS.filter(m => m.id === Number(marca))
9
10    console.log(nombreMarca);

```

Resultado: obtenemos un objeto con id y nombre.

```
▼ [{-}] ⓘ Resultado.jsx:10
  ▶ 0: {id: 1, nombre: 'Europeo'}
    length: 1
  ▶ [[Prototype]]: Array(0)
```

Podemos desestructurar para obtener sólo el nombre de la marca:

```
8      const [nombreMarca] = MARCAS.filter(m => m.id === Number(marca))
9
10     console.log(nombreMarca);
11
```

Obtenemos un objeto con el nombre de la marca:

```
▼ {id: 1, nombre: 'Europeo'} ⓘ Resultado.jsx:10
  id: 1
  nombre: "Europeo"
  ▶ [[Prototype]]: Object
```

De este objeto nos interesa la parte del nombre:

```
17      <h2 className="text-gray-700 font-black text-3xl">
18      |   Resumen
19      </h2>
20      <p>
21      |   <span className="font-bold">Marca:</span>{nombreMarca.nombre}
22      </p>
23      </div>
```

Resultado:

Resumen

Marca:Europeo

Mostramos el resto de los campos:

```
--      --
20      <p className="my-2">
21      |   <span className="font-bold">Marca:</span>{nombreMarca.nombre}
22      </p>
23      <p className="my-2">
24      |   <span className="font-bold">Plan:</span>{plan}
25      </p>
--      --
```

Así:

Resumen

Marca:Europeo

Plan:2

Pero queremos saber cuál es el plan 2:

```

8      const [nombreMarca] = MARCAS.filter(m => m.id === Number(marca))
9      const [nombrePlan] = PLANES.filter(p=> p.id === Number(plan))
10
11     console.log(nombreMarca);
12     console.log(nombrePlan);
13
14     if ( resultado === 0 ) return null
15
16     return (
17       <div className="bg-gray-100 text-center mt-5 p-5 shadow">
18         <h2 className="text-gray-700 font-black text-3xl">
19           Resumen
20         </h2>
21         <p className="my-2">
22           <span className="font-bold">Marca:</span>{nombreMarca.nombre}
23         </p>
24         <p className="my-2">
25           <span className="font-bold">Plan:</span>{nombrePlan.nombre}
26         </p>
27     )

```

Probamos:

MARCA

Europeo

AÑO

2016

ELIGE UN PLAN

Básico ☐ Completo ☒

COTIZAR

Resumen

Marca:Europeo

Plan:Completo

El año del coche:

```

25     <p className="my-2">
26       <span className="font-bold">Año:</span>{year}
27     </p>

```

Resultado:

Resumen

Marca:Europeo

Plan:Completo

Año:2016

El total de la cotización:

```

29 | | <p className="my-2 text-2xl">
30 | |   <span className="font-bold">Total Seguro:</span>{resultado}
31 | | </p>
    |

```

Resultado:

MARCA

Europeo

AÑO

2016

ELIGE UN PLAN

Básico ☐ Completo ☒

COTIZAR

Resumen

Marca: Europeo

Plan: Completo

Año: 2016

Total Seguro: 319,80 €

Si quisiera saber cuál es el precio del básico europeo de 2016, cambio a plan Básico. Por efecto del state que actualiza automáticamente los valores del form, nos pone en la vista el nombre del plan como Básico pero sin variar el precio del seguro.

Resumen

Marca: Europeo

Plan: Básico

Año: 2016

Total Seguro: 319,80 €

Esto no lo debería hacer hasta que el usuario no le dé al botón cotizar. Para ello utilizaremos funciones useCallback y useRef.

14. Como utilizar useCallback y useRef.

La documentación oficial de useCallback:

<https://es.reactjs.org/docs/hooks-reference.html#usecallback>

Devuelve un callback `memorizado`.

Pasa un callback en línea y un arreglo de dependencias. `useCallback` devolverá una versión memorizada del callback que solo cambia si una de las dependencias ha cambiado. Esto es útil cuando se transfieren callbacks a componentes hijos optimizados que dependen de la igualdad de referencia para evitar renders innecesarias (por ejemplo, `shouldComponentUpdate`).

`useCallback(fn, deps)` es igual a `useMemo(() => fn, deps)`.

Una versión memorizada del callback quiere decir según la Wikipedia:

Memoización

No debe confundirse con `Memorización`.

En **Informática**, el término **memoización** (del inglés *memoization*) es una técnica de **optimización** que se usa principalmente para acelerar los tiempos de cálculo, almacenando los resultados de la llamada a una **subrutina** en una **memoria intermedia o búfer** y devolviendo esos mismos valores cuando se llame de nuevo a la subrutina o función con los mismos parámetros de entrada.

Así, el `useCallback` nos va a permitir que no haga el render de la página hasta que le digamos cuando. Así en `Resultado.jsx`. Si quiero evitar que cuando cambie la marca, se actualice el resultado sin darle al botón Cotizar:

```

9   const [nombreMarca] = useCallback(
10     MARCAS.filter(m => m.id === Number(marca)),
11     [resultado]
12   )

```

No va a colocar el nombre de la marca cambiada a no ser que sea por resultado y éste es por efecto del botón cotizar. Probamos y vemos que aunque esté cambiando la marca, resultado espera por el botón:

MARCA

Americano

AÑO

2016

ELIGE UN PLAN

Básico

Completo

COTIZAR

Resumen

Marca: Europeo

Hacemos lo mismo para plan y año:

```

9      const [nombreMarca] = useCallback(
10        MARCAS.filter(m => m.id === Number(marca)),
11        [resultado]
12      )
13      const [nombrePlan] = useCallback(PLANES.filter(p=> p.id === Number(plan)),
14        [resultado]
15      )
    
```

Para el año tenemos que utilizar useRef:

`useRef` devuelve un objeto *ref* mutable cuya propiedad `.current` se inicializa con el argumento pasado (`initialValue`). El objeto devuelto se mantendrá persistente durante la vida completa del componente.

También, en `Resultado.jsx`:

```

src > components > Resultado.jsx > ...
1  import { useCallback, useRef } from 'react';

...

6  const { resultado, datos } = useCotizador();
7  const { marca, plan, year } = datos;
8  const yearRef = useRef(year);

...

25  <p className="my-2">
26    <span className="font-bold">Marca:</span>{nombreMarca.nombre}
27  </p>
28  <p className="my-2">
29    <span className="font-bold">Plan:</span>{nombrePlan.nombre}
30  </p>
31
32  <p className="my-2">
33    <span className="font-bold">Año:</span>
34    {yearRef.current}
35  </p>
    
```

Ahora aunque se cambie el año, hasta que no le demos al botón Cotizar no va cambiar ningún dato de Resumen. En principio está todo bien, salvo este warning, que es porque en `spinner.jsx` empleamos `class` en vez de `className` para referirnos a la clase:

```

Warning: Invalid DOM property `class`. Did you mean `className`?
    at div
    at div
    at Spinner
    at main
    at AppSeguro (http://localhost:3000/src/components/AppSeguro.jsx?t=1648566512434:27:7)
    at CotizadorProvider (http://localhost:3000/src/context/CotizadorProvider.jsx?t=164855510753:21:3)
    at App
    
```

Solución:

```

6  <div className="spinner">
7  <div className="rect1"></div>
    
```

15. Como utilizar useMemo y Fin de Proyecto.

Devuelve un valor **memorizado**.

Pasa una función de "crear" y un arreglo de dependencias. **useMemo** solo volverá a calcular el valor memorizado cuando una de las dependencias haya cambiado. Esta optimización ayuda a evitar cálculos costosos en cada render.

Recuerde que la función pasada a **useMemo** se ejecuta durante el renderizado. No hagas nada allí que normalmente no harías al renderizar. Por ejemplo, los efectos secundarios pertenecen a **useEffect**, no a **useMemo**.

```
src > components > Resultado.jsx > Resultado
1  import { useRef, useMemo } from 'react';
2  import useCotizador from '../hooks/useCotizador'
3  import { MARCAS, PLANES } from '../constants/index'
4
5  export const Resultado = () => {
6    const { resultado, datos } = useCotizador();
7    const { marca, plan, year } = datos;
8    const yearRef = useRef(year);
9
10   const [nombreMarca] = useMemo(() =>
11     MARCAS.filter(m => m.id === Number(marca)),
12     [resultado]
13   )
14   const [nombrePlan] = useMemo(() =>
15     PLANES.filter(p => p.id === Number(plan)),
16     [resultado]
17   )
18
19   if ( resultado === 0 ) return null
20 }
```

Esto hace la aplicación más rápida ya que evita cálculos costosos para cada render. En la consola de React se ve:



Search (text) 7 ^ v Resultado

App

- CotizadorProvider
 - Context.Provider
 - AppSeguro
 - Formulario
 - Resultado

props

new entry: ""

hooks

- Cotizador:
 - 1 Ref: "2020"
 - 2 Memo: [{...}]
 - 0: {id: 1, nombre: "Europeo"}
 - 3 Memo: [{...}]
 - 0: {id: 1, nombre: "Básico"}

rendered by