**Cotizador de Seguro de Automóvil con Context API**

Proyecto creado con VITE y Tailwind CSS

***¿Qué es Context API?***

Estado Global sin dependencias.

* Disponible desde React 16.3
* Puedes pasar el state o funciones desde el componente principal hasta los hijos, sin necesidad de pasarlo por cada componente.
* Cuenta con un hook llamado “useContext”.
* También se puede actualizar el state desde el hijo (o ejecutar una función que lo actualice).
* Si deseas crear librerías para React, Context es la forma de hacerlo.

1. Creamos el proyecto “npm create vite@latest”
2. En otra terminal, nos posicionamos sobre la ruta del proyecto e instalamos tailwind postcss y autoprefixer: “npm i tailwindcss postcss autoprefixer”.
3. Luego ejecutamos el comando “npx tailwindcss init –p” para crear el archivo de configuración de tailwind y el archivo de configuración de postCss.
4. Configuramos el archivo “tailwind.config.js”:

/\*\* *@type* {import('tailwindcss').Config} \*/

export default {

  content: ["index.html", "./src/\*\*/\*.jsx"],

  theme: {

    extend: {},

  },

  plugins: [],

}

1. Eliminamos todo lo que contenga el “index.css” y agregamos lo siguiente:

@tailwind *base*;

@tailwind *components*;

@tailwind *utilities*;

1. Sobre src creamos las carpetas:

* Components
* Constants
* Context
* Hooks
* Styles

1. Sobre components creamos “AppSeguro.jsx”

Importamos el AppSeguro.jsx en el App.jsx y lo renderizamos.

1. Sobre components creamos “Formulario.jsx” y lo importamos y renderizamos en el “AppSeguro.jsx”.
2. Sobre constants creamos un archivo “index.js”, en el cual vamos a tener un arreglo de las marcas de seguros, un arreglo de los planes disponibles y un arreglo de los años.

export *const* MARCAS = [

    {id: 1, nombre: "Europeo"},

    {id: 2, nombre: "Americano"},

    {id: 3, nombre: "Aciatico"}

];

*const* YEARMAX = new Date().getFullYear();

//Creamos una lista de los ultimos 20 años

export *const* YEARS = Array.from(new Array(20), (*valor*, *index*) *=>* YEARMAX - index) //Array.from convierte lo que le pasemos a un arreglo

export *const* PLANES = [

    {id: 1, nombre:"Básico"},

    {id: 2, nombre: "Completo"}

]

1. Importamos estas constante en el “Formulario.jsx” y comenzamos a trabajar en este archivo:

import { Fragment } from "react"

import {MARCAS, YEARS, PLANES} from "../constants/index.js"

*function* Formulario() {

  return (

    <>

      <form>

        <div className="my-5">

            <label className="block mb-3 font-bold text-gray-400 uppercase">

                Marca

            </label>

            <select

                name="marca"

                className="w-full p-3 bg-white border border-gray-200"

            >

                <option value="">--Selecciona Marca--</option>

                {MARCAS.map((*marca*) *=>* (

                    <option

                        key={marca.id}

                        value={marca.id}

                    >{marca.nombre}</option>

                ))}

            </select>

        </div>

        <div className="my-5">

            <label className="block mb-3 font-bold text-gray-400 uppercase">

                Año

            </label>

            <select

                name="año"

                className="w-full p-3 bg-white border border-gray-200"

            >

                <option value="">--Selecciona el Año--</option>

                {YEARS.map((*year*) *=>* (

                    <option

                        key={year}

                        value={year}

                    >{year}</option>

                ))}

            </select>

        </div>

        <div className="my-5">

            <label className="block mb-3 font-bold text-gray-400 uppercase">

               Elige un Plan

            </label>

         <div className="flex gap-3 items-center">

            {PLANES.map((*plan*) *=>* (

                <*Fragment* key={plan.id}>

                  <label>{plan.nombre}</label>

                  <input

                    type="radio" /\*\*unicamente puede elegir una opcion \*/

                    name="plan"

                    value={plan.id}

                  />

                </*Fragment*>

            ))}

         </div>

        </div>

        <input

            type="submit"

            className="w-full bg-emerald-500 hover:bg-emerald-600

            transition-colors text-white cursor-pointer p-3 uppercase font-bold"

            value="Cotizar"

        />

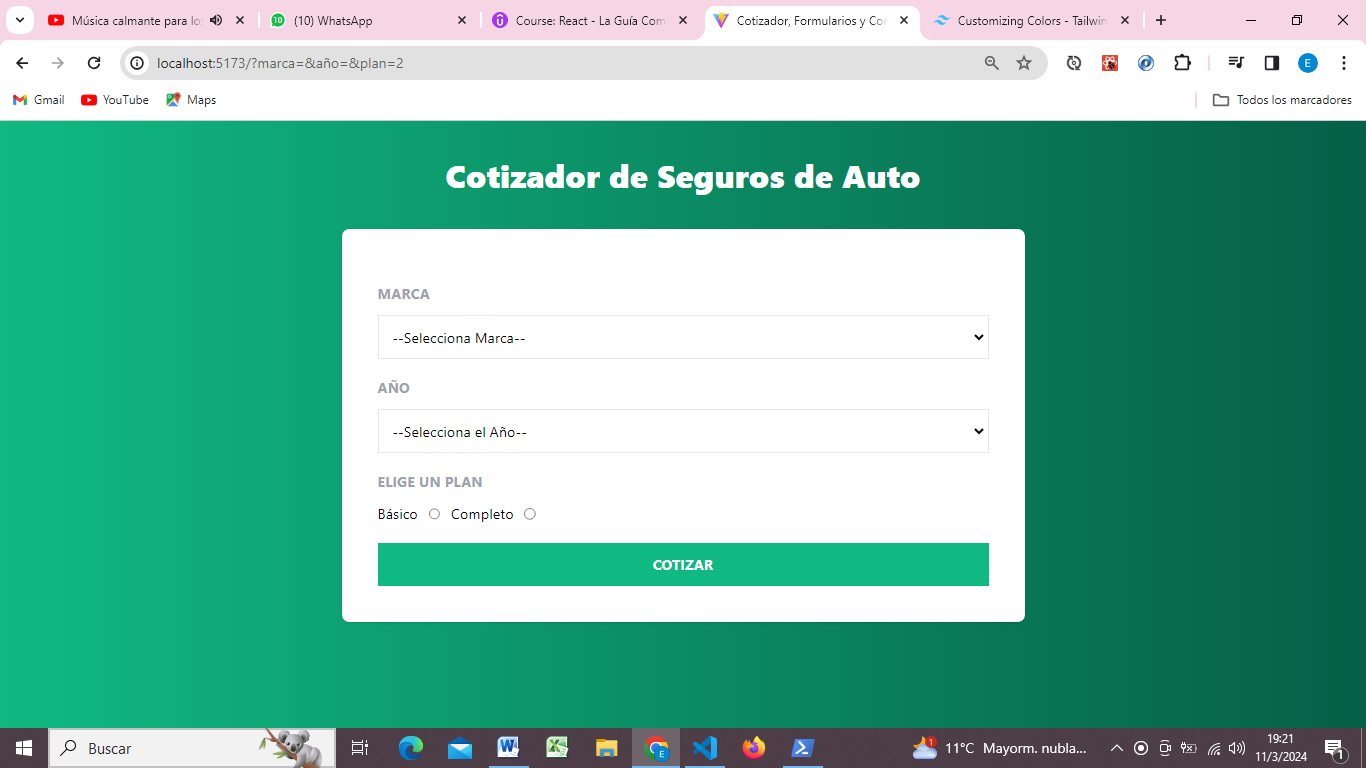
      </form>

    </>

  )

}

export default Formulario



1. Como próximo paso vamos a crear el “Context” y el “Provider”.

El ***PROVIDER***, es el lugar donde vamos a definir el state, podemos tener algunos effects, y tambien podemos crear funciones."

Provider es de donde nacen, de donde vienen los datos o cual es la fuente de los datos.

(el Provider es una función)

Para ello en la carpeta de context, vamos a crear un archivo “**Cotizador.provider.jsx**”.

**El Context y el Provider deben comenzar con la misma palabra:**

* CotizadorContext.
* CotizadorProvider.

Cotizador.provider.jsx:

import {createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

    return(

        <*CotizadorContext.Provider*>

            {children}

        </*CotizadorContext.Provider*>

    )

}

export {

    CotizadorProvider

}

export default CotizadorContext

* Children: son todos los componentes hijos de la aplicación.

1. Como siguiente paso vamos a importar el “CotizadorProvider” en el “App.jsx”.

El “CotizadorProvider” debe rodear la aplicación completa ya que si recordamos contiene “children” que son todos los componentes hijos de la aplicación.

De esta manera, los datos de CotizadorProvider, van a estar disponibles en todos los componentes de nuestra aplicación. Por ello es tan importante el children.

import {CotizadorProvider} from "./context/Cotizador.provider"

import AppSeguro from "./components/AppSeguro"

*function* App() {

  return (

    <>

      <*CotizadorProvider*>

        <*AppSeguro*/>

      </*CotizadorProvider*>

    </>

  )

}

export default App

1. Como pasar datos o funciones en el Provider:

Vamos a ver como crear datos dentro del Provider y consumirlos en otros componentes:

* Desde la llave de apertura hasta antes del return, podemos definir states, variables, effects, funciones propias, etc:
* Y dentro del “value” vamos a poner todo lo que deseemos poner a disposición y que se puede acceder en los demás componentes.

Cotizador.provider.jsx:

import {createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

    //Definimos variables, funciones, state, effect, etc

*const* hola = "Hola Mundo";

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

                hola: hola

            }}

        >

            {children}

        </*CotizadorContext.Provider*>

    )

}

export {

    CotizadorProvider

}

export default CotizadorContext

ahora, ¿Cómo hacemos para ac**ceder a la información que estamos haciendo disponible en el provider**?

En el componente que deseemos acceder a esta información, debemos importar el hook de “**useContext**” y debemos importar el context que hemos creado en el componente cotizador.provider.jsx (**CotizadorContext**).

En este caso, vamos acceder a la información desde el componente de formulario:

import { Fragment, useContext } from "react"

import {MARCAS, YEARS, PLANES} from "../constants/index.js"

import CotizadorContext from "../context/Cotizador.provider.jsx"

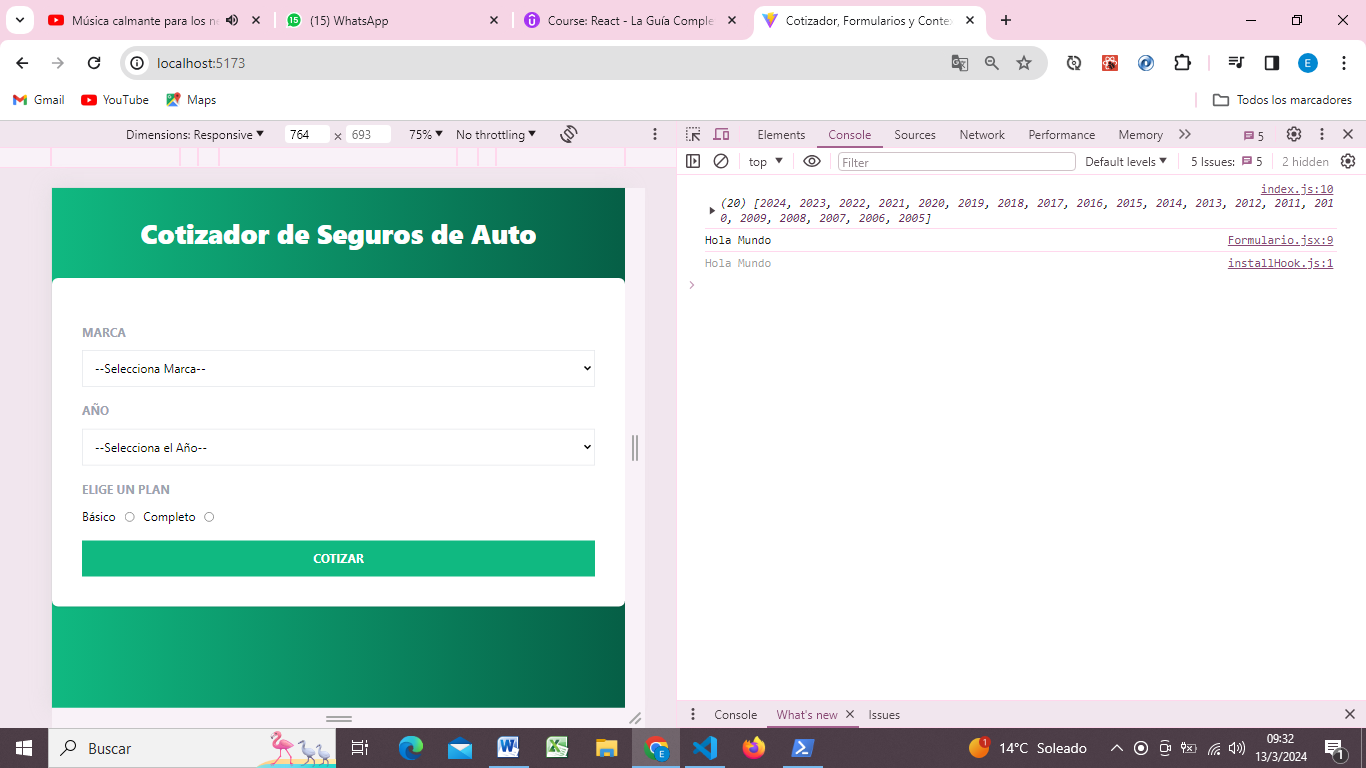
*function* Formulario() {

*const* {hola} = useContext(CotizadorContext)

    console.log(hola)

importamos el hook de useContext, y entre paréntesis debemos pasarle a que context es el que deseamos acceder, ya que podemos tener multiples context en nuestro proyecto.

Y luego podemos aplicar destructuring de objetos para acceder directamente a la información.



Vamos a poder acceder a esta información, únicamente en los componentes que estén dentro del CotizadorProvider en el App.jsx:

En este caso el AppSeguro, esta dentro de CotizadorProvider y Formulario está dentro de AppSeguro, entonces podemos acceder correctamente a la información:

<*CotizadorProvider*>

        <*AppSeguro*/>

</*CotizadorProvider*>

Pero si tuviéramos lo siguiente:

Si el formulario estuviera FUERA del cotizadorProvider, desde el no podríamos acceder a la información que hace disponible el provider.

<*CotizadorProvider*>

        <*AppSeguro*/>

</*CotizadorProvider*>

<*Formulario*/>

1. Pasar state desde el provider y consumirlo en componentes:

import { useState , createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

*const*[modal, setModal] = useState(false)

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

              modal,

              setModal

            }}

        >

            {children}

        </*CotizadorContext.Provider*>

    )

}

export {

    CotizadorProvider

}

export default CotizadorContext

*function* Formulario() {

*const* {setModal} = useContext(CotizadorContext)

  return (

    <>

        <button

            onClick={() *=>* setModal(true)}

        >

            Cambiar State

        </button>

Pasar el state funciona igual que pasar cualquier dato, variable, función, etc.

Otra manera en la que podemos modificar el state sin pasar específicamente el state, es pasar una función intermedia:

import { useState , createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

*const*[modal, setModal] = useState(false)

*const* cambiarState = () *=>* {

        setModal(!modal)

    }

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

             cambiarState

            }}

        >

            {children}

        </*CotizadorContext.Provider*>

    )

}

export {

    CotizadorProvider

}

export default CotizadorContext

import { Fragment, useContext } from "react"

import {MARCAS, YEARS, PLANES} from "../constants/index.js"

import CotizadorContext from "../context/Cotizador.provider.jsx"

*function* Formulario() {

*const* {cambiarState} = useContext(CotizadorContext)

  return (

    <>

        <button

            onClick={cambiarState}

        >

            Cambiar State

        </button>

1. Creando un Hook para acceder a los datos del Provider:

Ya vimos que para acceder a los datos del provider, primero debemos importar useContext, luego importar el context del que deseamos extraer los datos, luego pasarle el context a useContext, luego extraer el valor que deseemos utilizar y por ultimo hacer uso de este valor

Para simplificar todos estos pasos, podemos crear un Hook:

* Sobre la carpeta de “hooks” creamos un archivo llamado “useCotizador.jsx”.

Podemos tener un Hook para cada uno de los context que tengamos en nuestro proyecto.

import {useContext} from "react"

import CotizadorContext from "../context/Cotizador.provider"

*const* useCotizador = () *=>* {

    return useContext(CotizadorContext)

}

export default useCotizador

entonces ahora si queremos acceder a la información del provider desde el formulario, únicamente debemos importar este hook que hemos creado:

import { Fragment } from "react"

import {MARCAS, YEARS, PLANES} from "../constants/index.js"

import useCotizador from "../hooks/useCotizador.jsx"

*function* Formulario() {

*const* {cambiarState} = useCotizador()

  return (

    <>

        <button

            onClick={cambiarState}

        >

            Cambiar State

        </button>

1. VOLVIENDO AL PROYECTO, creando el State para el Formulario:

Debemos **colocar** **en** un **state**, **la marca** que seleccione el usuario en el formulario**, el año y el plan.**

**Esto lo vamos a hacer desde el provider, ya que debe pasar por diferentes componentes.**

Pero si fuese un state que requerimos únicamente en un componente, es mejor opción definirlo de manera local y no en un provider.

**Context.provider.jsx:**

En el Provider vamos a definir una función para modificar el state:

* Leemos el e.target.name, para saber que es lo que estamos seleccionando, si el plan, el año o el modelo.
* Y leemos el e.target.value, para saber cual fue el valor seleccionado para cada uno de los name.

*const* CotizadorProvider = ({*children*}) *=>* {

    //Función que actualiza el State

*const* handleChangeDatos = *e* *=>* {

        console.log(e.target.name)

        console.log(e.target.value)

    }

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

                handleChangeDatos

            }}

        >

**Formulario.jsx:**

Asociamos esta función a los select y al input:

<select

                name="marca"

                className="w-full p-3 bg-white border border-gray-200"

                onChange={ *e* *=>* handleChangeDatos(e)}

            >

<select

                name="year"

                className="w-full p-3 bg-white border border-gray-200"

                onChange={ *e* *=>* handleChangeDatos(e)}

            >

   <input

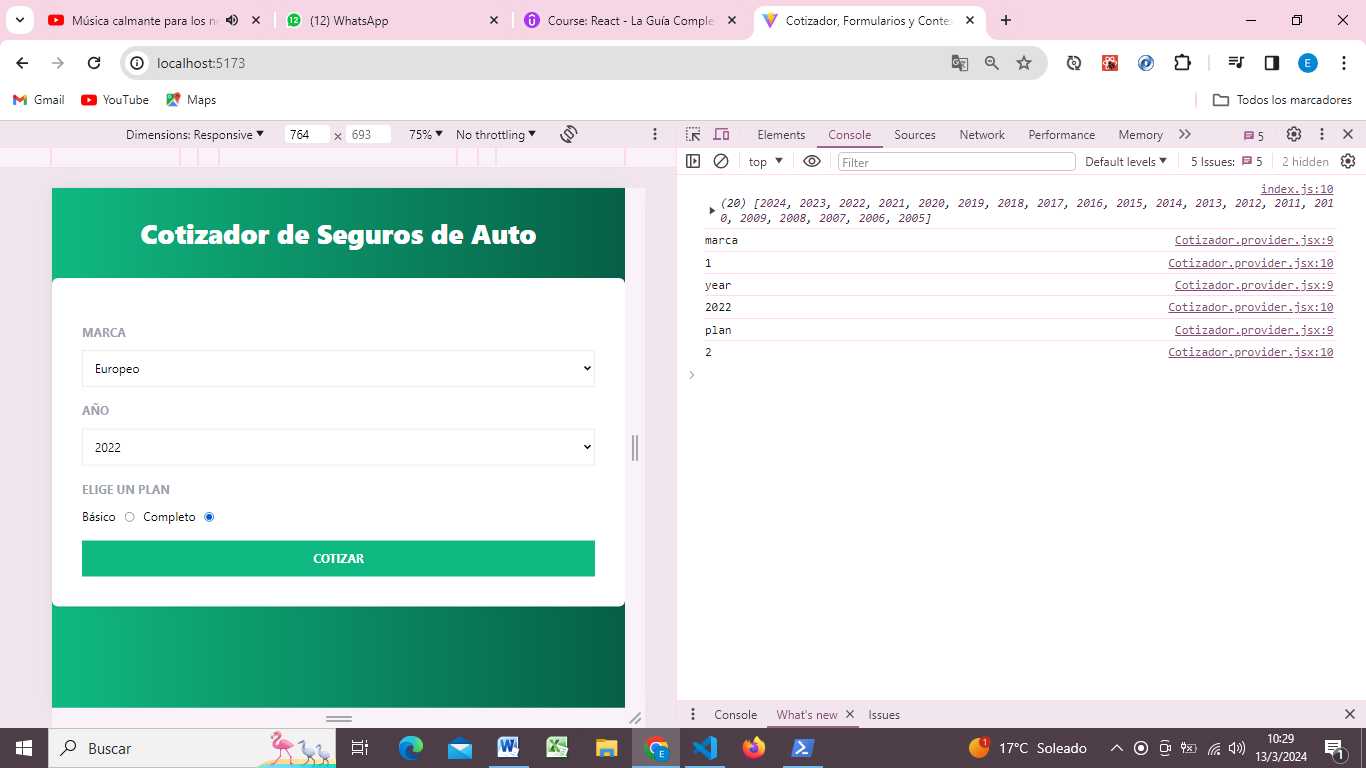
                    type="radio" /\*\*unicamente puede elegir una opcion \*/

                    name="plan"

                    value={plan.id}

                    onChange={ *e* *=>* handleChangeDatos(e)}

                  />



Cuando trabajamos con un **state** pero con un objeto dentro del State, lo ideal es ir tomando una copia de lo que hay almacenado en el state, y luego ir reescribiendo las propiedades:

import { useState, createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

*const* [datos, setDatos] = useState({

Objeto dentro del state.

Podemos hacerlo de esta manera o trabajar con las variables separadas

        marca:"",

        year:"",

        plan:""

    })

    //Función que actualiza el State

*const* handleChangeDatos = *e* *=>* {

        setDatos({

            ...datos, //tomamos una copia del state previo

            [e.target.name] : e.target.value //modificamos el state

        })

    }

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

Hacemos disponibles los datos

            datos,

            handleChangeDatos

            }}

        >

            {children}

        </*CotizadorContext.Provider*>

    )

}

export {

    CotizadorProvider

}

export default CotizadorContext

* En el Formulario.jsx, asociamos los datos al value de los selects:

    <select

                name="marca"

                className="w-full p-3 bg-white border border-gray-200"

                onChange={ *e* *=>* handleChangeDatos(e)}

                value={datos.marca}

            >

 <select

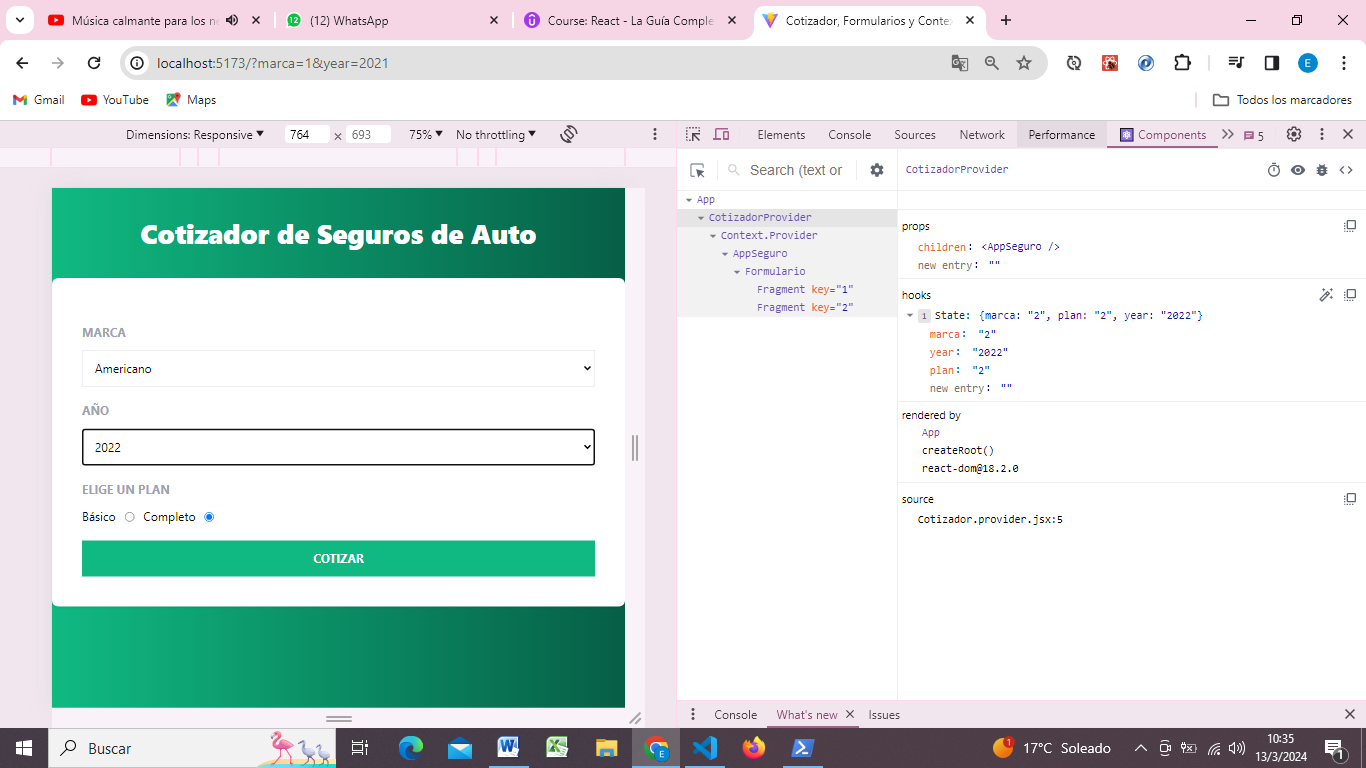
                name="year"

                className="w-full p-3 bg-white border border-gray-200"

                onChange={ *e* *=>* handleChangeDatos(e)}

                value={datos.year}

            >



1. El próximo paso, es VALIDAR EL FORMULARIO:

Debemos tener los 3 campos lleno para poder realizar la cotización. Para ello debemos realizar una validación previa a la cotización.

El evento del submit, únicamente lo requerimos en el componente de formulario. Para ello vamos a trabajar en Formulario.jsx.

Requerimos un state de Error, y este lo vamos a definir en el provider para que este disponible en todos los componentes.

Ademas, para mostrar el error, vamos a crear sobre “components” un componente llamado “Error.jsx”:

**Cotizador.provider.jsx**:

Definimos el state de error y lo hacemos disponible en todos los componentes:

import { useState, createContext} from "react"

*const* CotizadorContext = createContext()

*const* CotizadorProvider = ({*children*}) *=>* {

*const* [datos, setDatos] = useState({

        marca:"",

        year:"",

        plan:""

    })

*const* [error, setError] = useState("")

    //Función que actualiza el State

*const* handleChangeDatos = *e* *=>* {

        setDatos({

            ...datos, //tomamos una copia del state previo

            [e.target.name] : e.target.value //modificamos el state

        })

    }

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

            datos,

            setError,

            error,

            handleChangeDatos

            }}

        >

**Formulario.jsx:**

Creamos la función para validar el formulario y se le asociamos al Form.

import { Fragment } from "react"

import {MARCAS, YEARS, PLANES} from "../constants/index.js"

import useCotizador from "../hooks/useCotizador.jsx"

import Error from "./Error.jsx"

*function* Formulario() {

*const* {datos, setError, error, handleChangeDatos} = useCotizador()

*const* handleSubmit = *e* *=>* {

        e.preventDefault(); //prevenimos el envio del formulario, para primero realizar la validacion

        if(Object.values(datos).includes("")){

            setError("Todos los campos son obligatorios")

            return

        }

        setError("")

    }

  return (

    <>

    {/\*En caso de que error contenga algo,

    mostramos el componente de error\*/

    error && <*Error*/>

    }

     <form onSubmit={handleSubmit}>

**Error.jsx:**

Motramos el error en pantalla:

import useCotizador from "../hooks/useCotizador"

*function* Error() {

*const* {error} = useCotizador()

  return (

    <div className="bg-red-200 p-3 uppercase font-bold text-center border-l border-solid border-2 border-red-700">

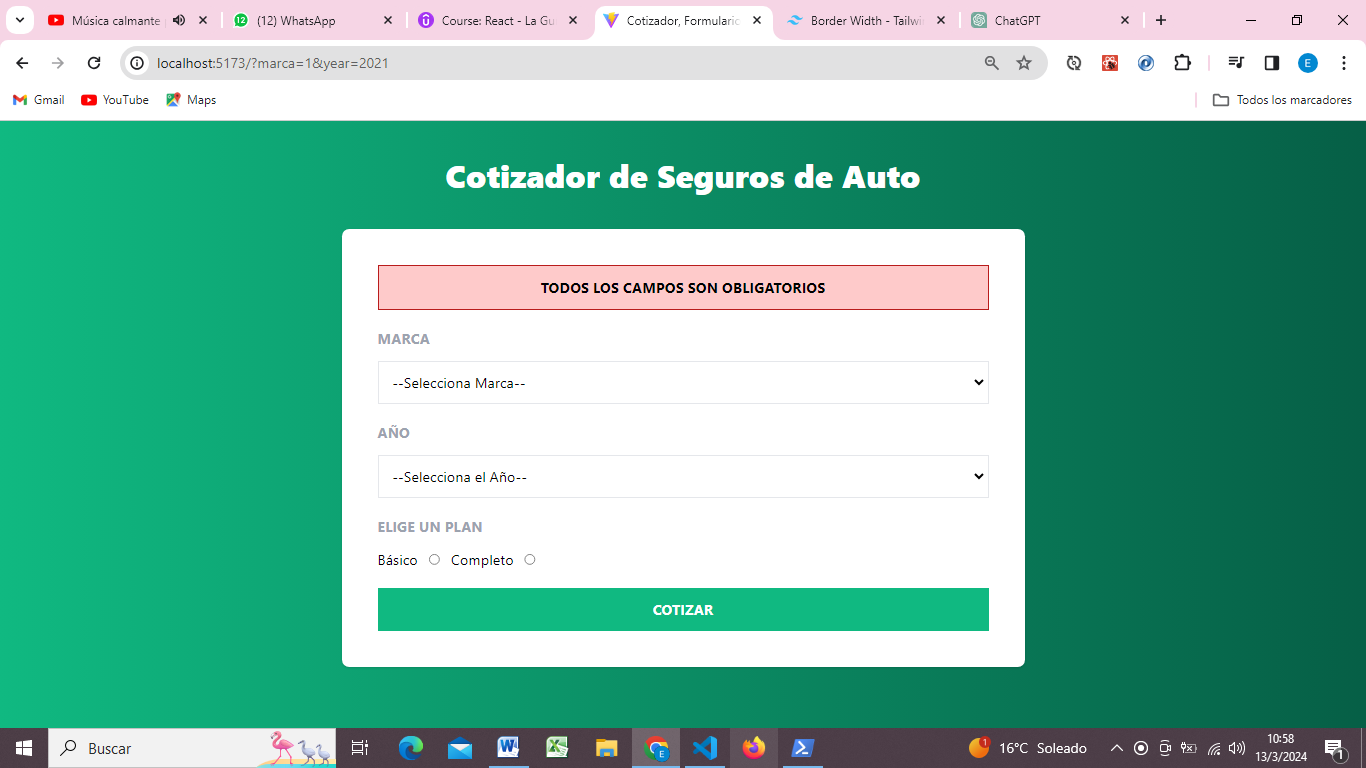
      {error}

    </div>

  )

}

export default Error



1. El próximo paso es crear la función para cotizar el seguro:

En Cotizador.provider.jsx, vamos a crear la función que se va a encargar de cotizar el seguro.

Tenemos tres marcas de auto:

* Europeo, es el más caro.
* Americano, es intermedio.
* Asiático, es el más barato.

El año del auto, mientras más actual sea el año, más caro es el seguro.

Al igual que el plan, el básico es el más barato y el completo es el más caro.

        //Una base

        //Debemos obtener diferencia de años.

        //Hay que restar el 3% por cada año (Es decir, un seguro 2020 es un 3% más barato que un seguro 2021)

Marcas:

        //Americano incrementa el costo el 15%

        //Europeo incrementa el costo el 30%

        //Asiático incrementa el costo el 5%

Planes:

        //Plan básico incrementa 20%

        //Plan completo incrementa 50%

Debemos realizar muchas operaciones, y para ello nos vamos apoyar de helpers.

1. Trabajando con el Algoritmo de Cotización:

Sobre src creamos una carpeta “**helpers**”. Y dentro de ella un archivo “**index.jsx**”.

Dentro del index.js vamos a tener las funciones

export *function* obtenerDiferenciaYear(*year*){

    return new Date().getFullYear() - year;

}

//La opcion 1 o la opcion 2 son correctas

/\* OPCION 1

export function calcularMarca(marca, resultado){

    let incremento

    if(marca === "1"){

        incremento = (30 \* resultado) / 100;

        return incremento

    }

    else if (marca === "2"){

        incremento = (15 \* resultado) / 100;

        return incremento

    }

    else if(marca === "3"){

        incremento = (5 \* resultado) / 100

        return incremento

    }

}

\*/

//OPCION 2

export *function* calcularMarca(*marca*){

*let* incremento

    if(marca === "1"){

        incremento = 1.30

        return incremento

    }

    else if (marca === "2"){

        incremento = 1.15

        return incremento

    }

    else if(marca === "3"){

        incremento = 1.05

        return incremento

    }

}

//Calcular PLAN

export *function* calcularPlan(*plan*){

*let* incremento

    if(plan === "1"){ //id Plan basico

        incremento = 1.20

      return incremento

    }

    else{

        incremento = 1.50 //id Plan completo

        return incremento

    }

}

//Formatear Dinero

export *function* formatearDinero(*cantidad*){

    return cantidad.toLocaleString("es-ES", {

        style: "currency",

        currency: "EUR",

        currencyDisplay: "symbol" // Mostrar el símbolo de la moneda primero

    })

}

**Cotizador.provider.jsx**:

Vamos a crear un state en el que vamos almacenar el resultado:

*const* [resultado, setResultado] = useState(0)

    //Funcion que cotiza el seguro

*const* cotizarSeguro = () *=>* {

        //Una base

*let* resultado = 2000

        //Obtener diferencia de años (año actual y año seleccionado por el usuario)

*const* diferencia = obtenerDiferenciaYear(datos.year)

        //Hay que restar el 3% por cada año(un seguro 2020 es un 3% mas barato q un seguro 2021)

        /\*\*Restamos el 3 por ciento por cada año, del resultado \*/

        resultado -= ((diferencia \* 3) \* resultado) / 100;

        //Americano incrementa el costo el 15%

        //Europeo incrementa el costo el 30%

        //Asiatico incrementa el costo el 5%

       /\*resultado = resultado + calcularMarca(datos.marca, resultado) OPCION 1\*/

        resultado \*= calcularMarca(datos.marca) //OPCION 2

        //Plan básico incrementa 20%

       //Plan completo incrementa 50%

       resultado \*= calcularPlan(datos.plan)

       resultado = formatearDinero(resultado)

       setResultado(resultado)

    }

    return(

        <*CotizadorContext.Provider*

            value={{ /\*\*hacemos disponibles los datos que deseemos en los demas componentes \*/

            datos,

            setError,

            error,

            handleChangeDatos,

            cotizarSeguro

            }}

        >

            {children}

        </*CotizadorContext.Provider*>

    )

}

Y luego en formulario.jsx, una vez que el formulario pasa la validación, llamamos la función “cotizarSeguro”:

*const* handleSubmit = *e* *=>* {

        e.preventDefault(); //prevenimos el envio del formulario, para primero realizar la validacion

        if(Object.values(datos).includes("")){

            setError("Todos los campos son obligatorios")

            return

        }

        setError("")

    cotizarSeguro()

    }

1. El próximo paso es mostrar el Total de la cotización.

Esto lo vamos hacer en AppSeguro.jsx:

Vamos añadir un spinner de carga mientras se realiza la cotización. (<https://tobiasahlin.com/spinkit/> )

En primer lugar, en “Cotizador.provider.jsx” creamos un nuevo state para agregar un spinner.

*const* [cargando, setCargando] = useState(false)

Hacemos disponible este valor, y lo extraemos en AppSeguro.jsx:

*const*{cargando} = useCotizador()

 {cargando ? <*Spinner*/> : <*Resultado*/>}  {/\*\*si cargando está como true, se muestra el Spinner, si esta como false, se muestra el resultado \*/}

Creamos sobre components, un archivo llamado “Spinner.jsx”. Agregamos el código html, y sobre styles creamos un archivo Spinner.css para agregar el código css.

**Spinner.jsx:**

import  "../styles/Spinner.css"

*function* Spinner() {

  return (

      <div className="spinner">

  <div className="double-bounce1"></div>

  <div className="double-bounce2"></div>

</div>

  )

}

export default Spinner

**Spinner.css:**

.spinner {

*width*: 40px;

*height*: 40px;

*position*: relative;

*margin*: 100px auto;

  }

  .double-bounce1, .double-bounce2 {

*width*: 100%;

*height*: 100%;

*border-radius*: 50%;

*background-color*: #098346;

*opacity*: 0.6;

*position*: absolute;

*top*: 0;

*left*: 0;

*-webkit-animation*: sk-bounce 2.0s infinite ease-in-out;

*animation*: sk-bounce 2.0s infinite ease-in-out;

  }

  .double-bounce2 {

*-webkit-animation-delay*: -1.0s;

*animation-delay*: -1.0s;

  }

  @-webkit-keyframes *sk-bounce* {

    0%, 100% { *-webkit-transform*: scale(0.0) }

    50% { *-webkit-transform*: scale(1.0) }

  }

  @keyframes *sk-bounce* {

    0%, 100% {

*transform*: scale(0.0);

*-webkit-transform*: scale(0.0);

    } 50% {

*transform*: scale(1.0);

*-webkit-transform*: scale(1.0);

    }

  }

Y creamos un componente “Resultado.jsx” para mostrar el resultado:

import useCotizador from "../hooks/useCotizador"

*function* Resultado() {

*const*{resultado} = useCotizador()

    if(resultado === 0) return null //esto para q cuando resultado este en 0, no se muestre el 0 en pantalla

  return (

    <div>

      {resultado}

    </div>

  )

}

export default Resultado

1. Como siguiente paso, vamos añadir más información al resultado:

En el componente de Resultado.jsx, vamos a mostrar un resumen con los valores que hayamos seleccionado.

Si recordamos, la marca la evaluábamos por un id, pero si mostramos ese valor de id en pantalla, el cliente no va a entender. Pero si recordamos, el objeto de marcas también contaba con el nombre.

Entonces lo que vamos hacer es, utilizando el método FILTER (array method) vamos a comparar la variable que estamos seleccionando con las que hayan en el arreglo, el valor del arreglo que sea igual a la variable que estamos seleccionando, se va almacenar en una nueva variable.

Le vamos aplicar array destructuring a esa variable y vamos a acceder a la propiedad de nombre para poder mostrar en pantalla:

import useCotizador from "../hooks/useCotizador"

import {MARCAS, PLANES} from "../constants"

Debemos importar el arreglo de MARCAS, para poder filtrar sobre el mediante el método filter.

*function* Resultado() {

*const*{resultado, datos} = useCotizador()

*const*{marca, year, plan} = datos //aplicamos object destructuring a datos

*const* [nombreMarca] = MARCAS.filter(*m* *=>* m.id === Number(marca))

    if(resultado === 0) return null //esto para q cuando resultado este en 0, no se muestre el 0 en pantalla

  return (

    <div className="bg-grey-100 text-center p-5 shadow">

        <h2 className="text-gray-600 font-black text-3xl">

            Resumen

        </h2>

        <p className="my-2">

            <span className="font-bold">Marca:</span>

            {nombreMarca.nombre}

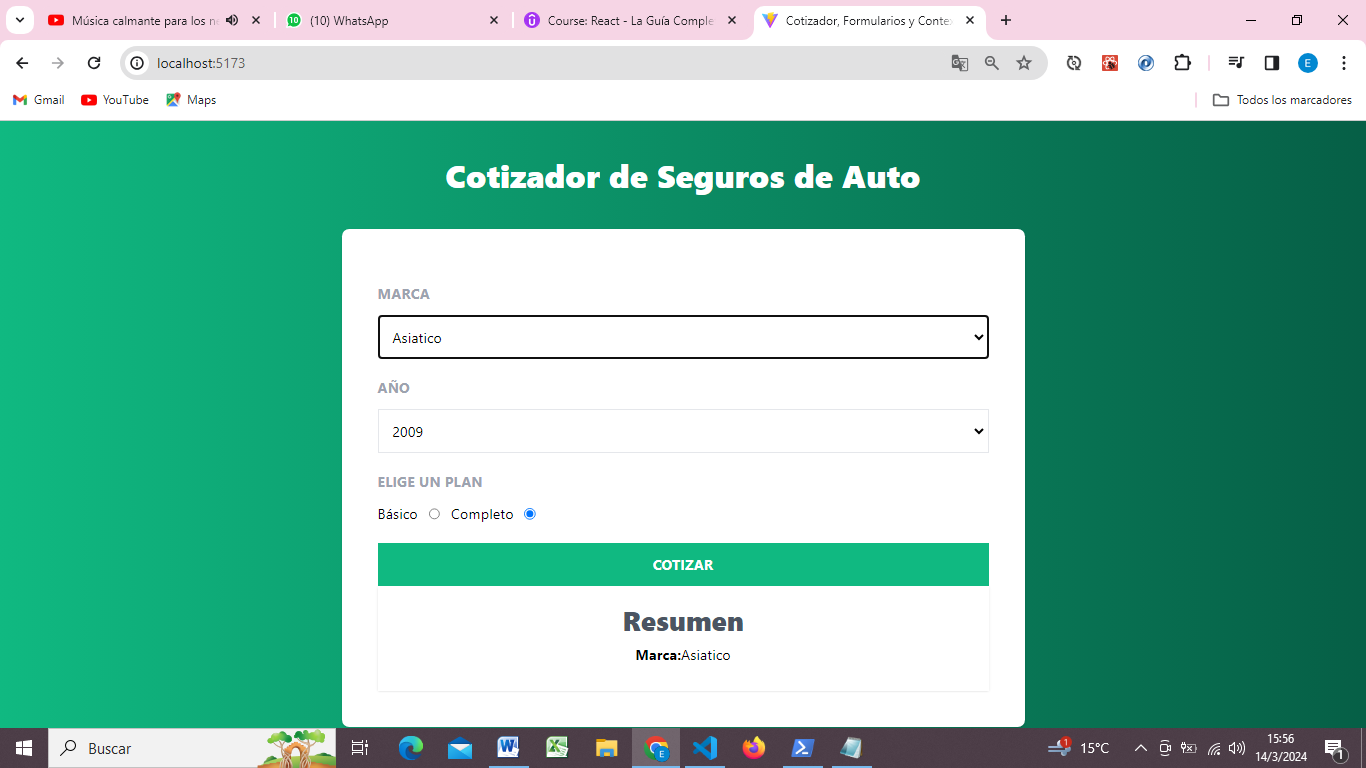
        </p>

    </div>

  )

}

export default Resultado



* Para mostrar el nombre del plan, hacemos exactamente lo mismo que con las marcas:

import useCotizador from "../hooks/useCotizador"

import {MARCAS, PLANES} from "../constants"

*function* Resultado() {

*const*{resultado, datos} = useCotizador()

*const*{marca, year, plan} = datos

*const* [nombreMarca] = MARCAS.filter(*m* *=>* m.id === Number(marca))

*const* [nombrePlan] = PLANES.filter(*p* *=>* p.id === Number(plan))

    if(resultado === 0) return null //esto para q cuando resultado este en 0, no se muestre el 0 en pantalla

  return (

    <div className="bg-grey-100 text-center p-5 shadow">

        <h2 className="text-gray-600 font-black text-3xl">

            Resumen

        </h2>

        <p className="my-2">

            <span className="font-bold">Marca: </span>

            {nombreMarca.nombre}

        </p>

        <p className="my-2">

            <span className="font-bold">Plan: </span>

            {nombrePlan.nombre}

        </p>

        <p className="my-2">

            <span className="font-bold">Año del Auto: </span>

            {year}

        </p>

        <p className="my-2 text-2xl">

            <span className="font-bold">Total Cotización: </span>

            {resultado}

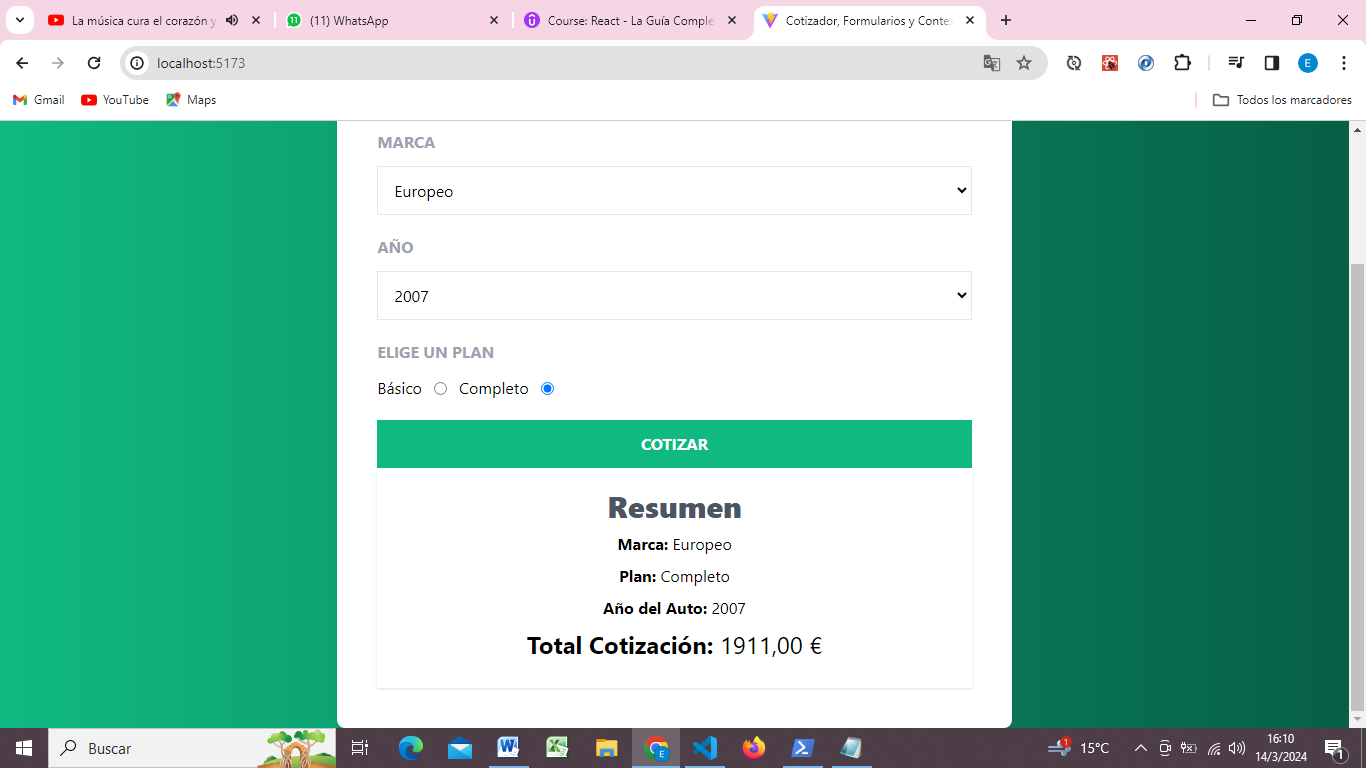
        </p>

    </div>

  )

}

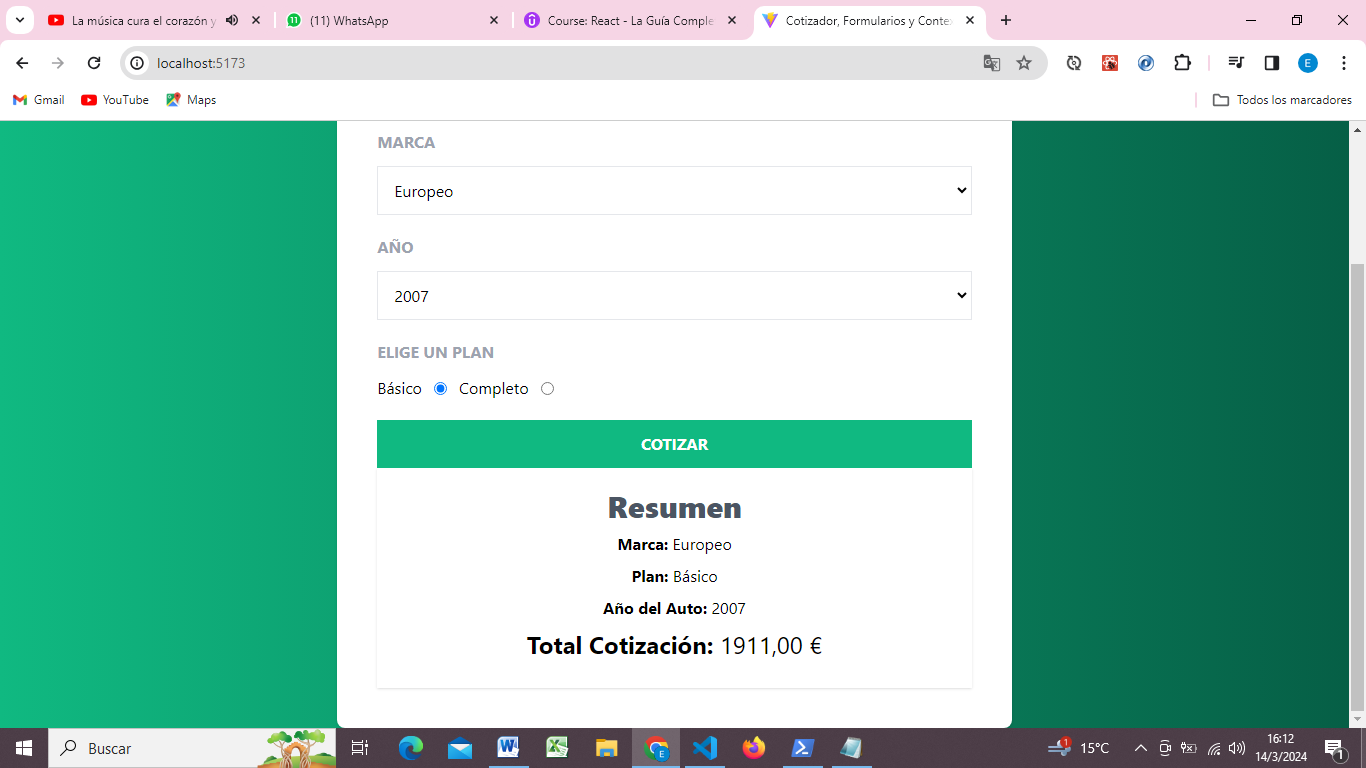
export default Resultado



Tenemos un PROBLEMA:

Como podemos ver en la imagen, tenemos el plan COMPLETO con el total de 1911,00euros.

Pero si selecciono el otro plan, podemos ver que se actualiza el plan, pero el precio sigue estando igual:



Lo mismo pasa con los años y la marca, esos valores se actualizan pero el total sigue siendo el mismo.

Esto es debido al re render de react, pero vamos hacer lo siguiente:

Vamos hacer que hasta que no se vuelva a presionar el botón de “Cotizar” los valores NO van a cambiar.

1. Como utilizar useCallback y useRef

**useCALLBACK y useREF**

* ***useCallback:***

Devuelve un callback memorizado.

En computación, memorizado es una técnica de optimización utilizado principalmente para acelerar los programas de computadoras, almacenando los resultados de llamados o de funciones que consumen muchos recursos y retorna la versión cacheada cuando la entrada de datos ocurre de nuevo.

useCallack, debe tener una función y un arreglo de dependencias.

Cuando resultado cambie, se aplica el re render:

import {useCallback, useMemo} from "react"

*function* Resultado() {

*const* [nombreMarca] = useCallback(

FUNCIÓN

MARCAS.filter(*m* *=>* m.id === Number(marca)),

[resultado]

ARREGLO DE DEPDENCIAS

)

*const* [nombrePlan] = useCallback(PLANES.filter(*p* *=>* p.id === Number(plan)), [resultado])

* useRef:

Devuelve un objeto REF mutable.

El objeto devuelto se mantendrá persistente durante la vida completa del componente.

De esta manera, se “congela” el valor hasta que el componente cambie.

*const* yearRef = useRef(year)

   <p className="my-2">

            <span className="font-bold">Año del Auto: </span>

            {yearRef.current}

        </p>