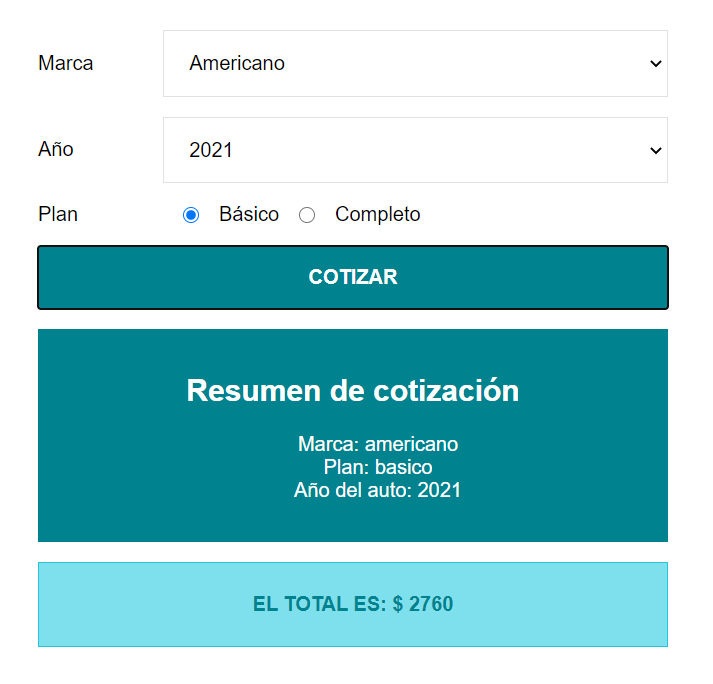
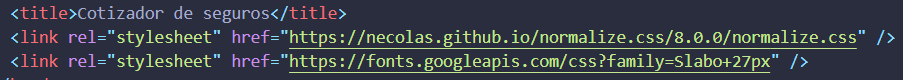
**Proyecto 4: cotizador de seguros**

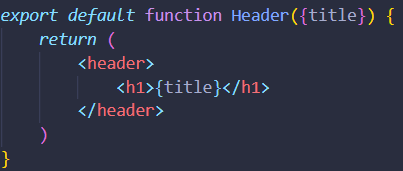
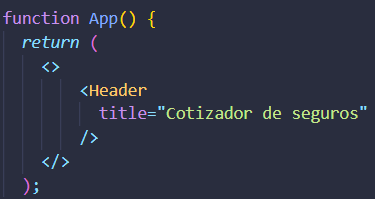
**Resultado Final:**

**1.** **Eliminamos** los archivos de **testing**, el **logo** que viene por defecto y el archivo **app.css**.

**2**. Vamos al archivo **index.html** (public) y en el <**head**> agregamos las dos líneas de **código** que nos pasan en el enlace de **gitHub**.

**3**. En el **index.css** pegamos el **código** que nos pasan en **gitHub**.

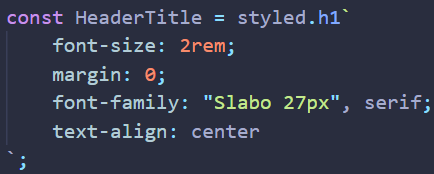
**4**. Instalamos la librería [Emotion](https://emotion.sh/docs/introduction), para manejar los style-components, usando: **npm i @emotion/styled @emotion/React**

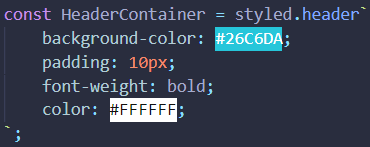
**5. Creamos** el componente **Header.js**, que va a contener una etiqueta de <header> y un <h1>. Luego lo importamos en App.js

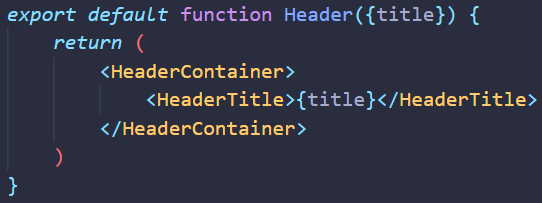
**6**. **Usando styled-components**: en **Header.js** **importamos** la librería **Emotion**, y empezamos a **crear los styled-components**. Por el momento vamos a crear 2: uno para la etiqueta <**header**> y otro para el <**h1**>

**Sintaxis**: creamos una **constante** con un nombre determinado, la **igualamos a styled.x** (nombre de la etiqueta), abrimos **strings literals** y colocamos el **código CSS**. Luego **reemplazamos** la **etiqueta** que se retorna por el nombre de la **constante creada**.

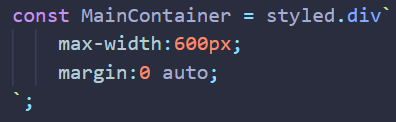
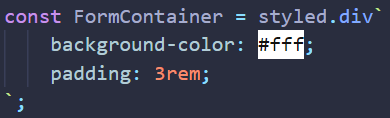
**<header>**: creamos una constante con un nombre que se relacione (**HeaderContainer**) y la igualamos a **styled.header**, luego abrimos los **string literals**, y colocamos el **código CSS** deseado.

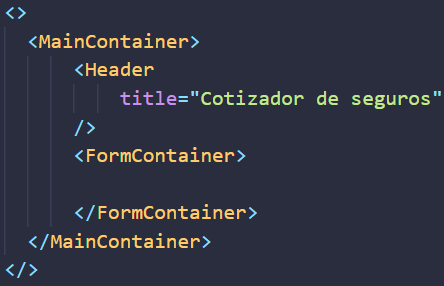
<**h1**>: creamos una constante con un nombre que se relacione (**HeaderTitle**) y la igualamos a **styled.header**, luego abrimos los **string literals**, y colocamos el **código CSS deseado**.

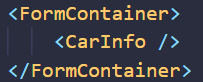


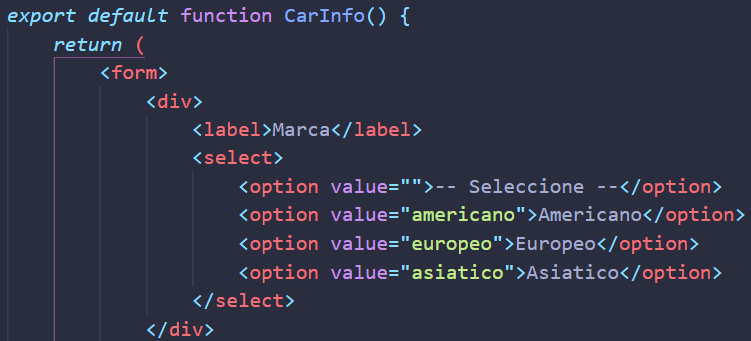


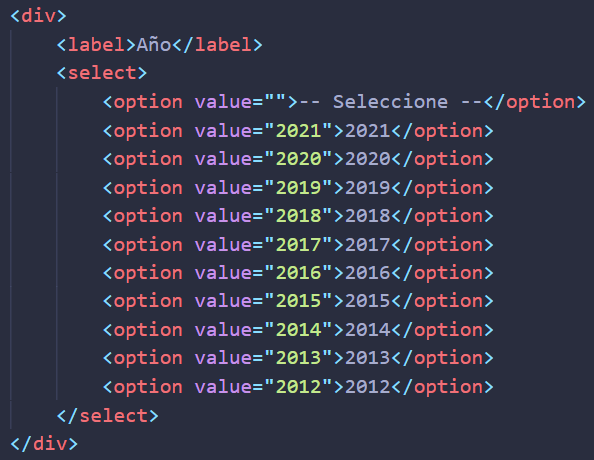
**7**. Ahora vamos a **crear** un **contenedor** **general** y **uno** **para** el **formulario** que vamos a hacer más adelante. Para esto, **dentro** de **App.js** creamos el **styled-component** **MainContainer** y **FormContainer.**

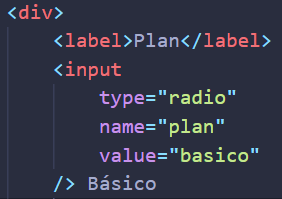
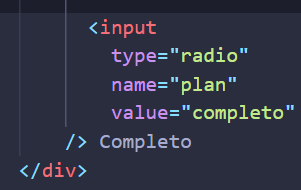


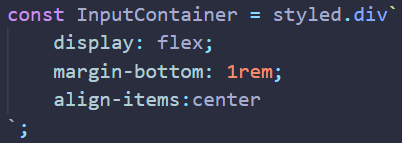


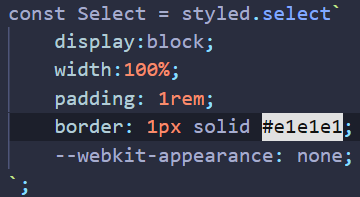
**8**. Creamos el componente **CarInfo.js** que va a ser un **formulario** donde el usuario va a **ingresar** la **marca del auto**, el **año** y el tipo de **plan** que quiere. Es importante recordar que en los input “radio” el name debe ser el mismo en ambos. Finalmente **importamos** el **componente** dentro de **App.js**

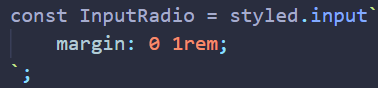
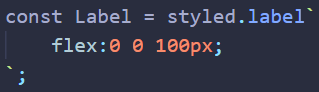


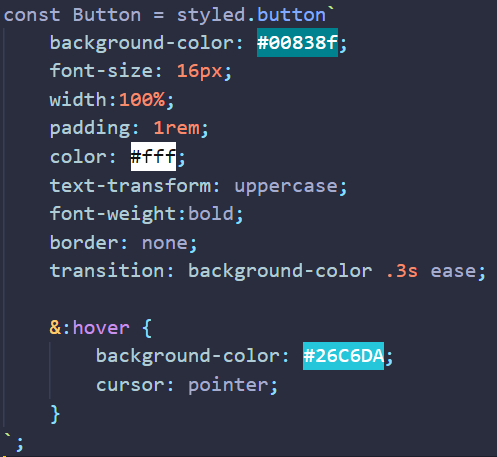


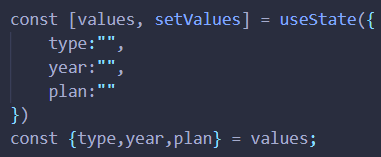
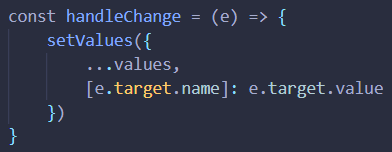


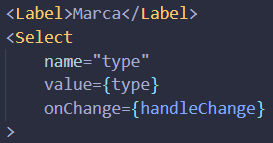
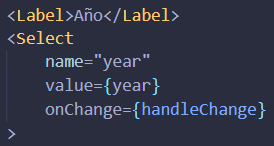
**9**. Creamos los **styled-components** para los elementos del **formulario**: **InputContainer** (div), **Label** (label), **Select** (select), **InputRadio** (input), **Button** (button).



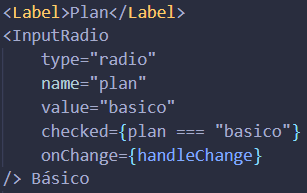
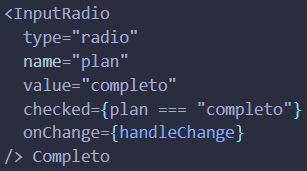




**10**. Ahora vamos a **guardar** en un **estado** los **datos** de: **marca, año, y plan**. Para esto creamos el estado value y le damos el **estado inicial** de un **objeto literal** con 3 **propiedades** que contienen un **string vacío**. Luego ponemos en variables cada propiedad para que sea más fácil utilizarlas. **Finalmente** creamos la **función** que va a **guardar** los **datos** de los **inputs** en el estado.

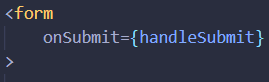
**11**. A los **select** le ponemos el **mismo** **name** y **value** que la **propiedad** del **objeto literal**, y en el evento **Onchange** la **funcion** creada anteriormente. El value es para que se mantengan los valores seleccionados luego de cotizar.

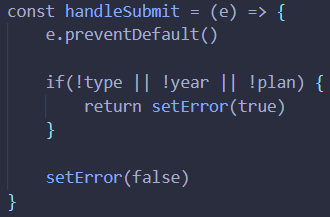
**12**. En los **inputs** tenemos que **agregar** dos **propiedades**: en el evento **onChange** la funcion y en “**checked**” un **condicional** diciendo que si la **propiedad** “**plan**” **del objeto litral** es **igual** al **name** del **input**, esa **opcion** **queda seleccionada**.



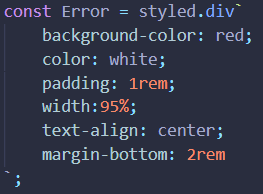
**13**. **Validando los campos**: creamos un **estado** para los **errores** y lo iniciamos en **false**. Luego hacemos una **función** que va a manejar el **submit** del formulario, allí evitamos que se envié y validamos los datos. En caso de que **allá error cambiamos el estado a true** y **retornamos la función**, de lo contrario ponemos el estado en false.





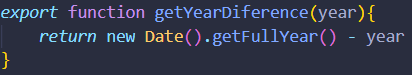


**14**. **Mostrando los errores**: creamos un styled-component para desplegar el error (es un div) y luego **preguntamos si error es true**, **en caso de que si imprimimos un mensaje**.

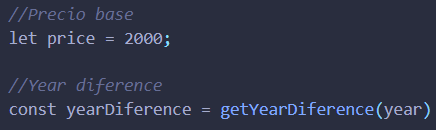




**15**. **Calculando el precio del seguro**: para determinar el precio del seguro tenemos que tener en cuenta el año del mismo, la marca y el plan.

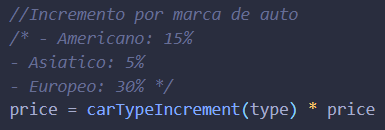
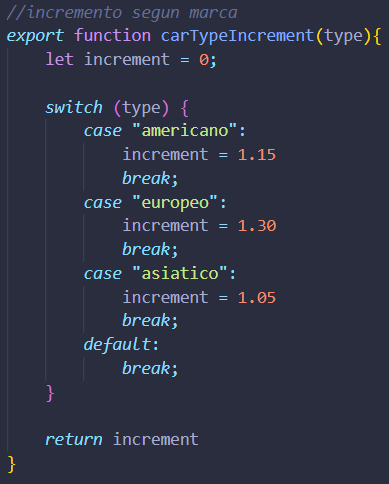
**Según año**: para esto primero debemos **obtener** la **diferencia** de **año** que tiene el **auto** **con** el **año actual**, es por eso que vamos a **crear** un **helper** que se va a encargar de hacer la cuenta. Recibe un año y le resta el año actual.

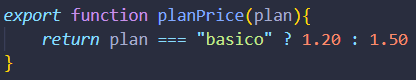
Dentro de **CarInfo.js**, en el handleSubmit, **definimos** un **precio base** del seguro (**2000**), **luego** obtenemos la **diferencia de años** que hay y **finalmente** le **cambiamos** el **valor** del **precio base** haciendo la siguiente cuenta: **por cada año de diferencia que haya, se le resta un 3% del valor base.**





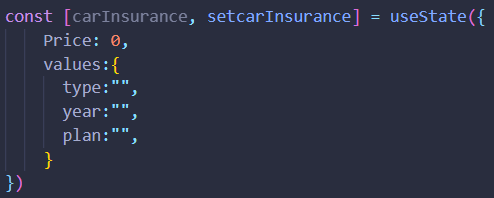
**Según marca**: el **precio base** (que ya fue afectado por el año) va a tener su **valor incrementado** **según la marca** seleccionada (Americano 15%, Asiático 5% y Europeo 30%). Para esto vamos a **crearle** otra **función al helper** que mediante un **switch** va a **determinar** el **incremento adecuado**. Finalmente dentro de **CarInfo.js** lo requerimos y utilizamos

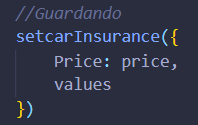


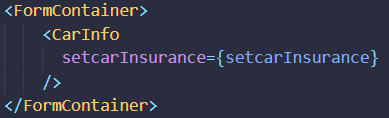
**Según Plan**: el **precio base** (que ya fue afectado por al año y la marca) va a **aumentar** según el **plan seleccionado** (básico 20%, completo 50%). Para esto creamos otra función dentro del **helper**, que va a recibir un plan y en base a cual sea va a retornar el monto que se debe incrementar.

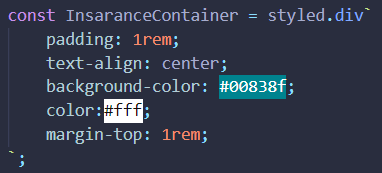
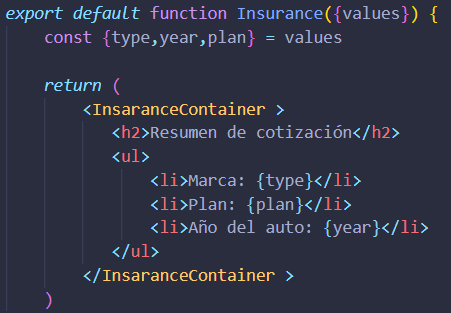
Luego dentro de Carinfo.js, le aplicamos la función al precio base.

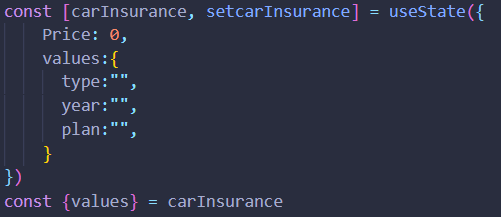
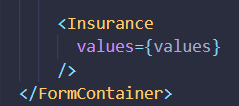
**16**. **Guardando los datos**: debemos crear un **nuevo estado** dentro de **App.js**, y le vamos a asignar un **objeto literal** con: el **total de la cotización**, los **datos de la cotización** (que a su vez va a tener un **objeto literal** con la **marca**, el **modelo** y el **año**). Podríamos dejar el objeto vacío y que se auto complete, pero lo hacemos de esta forma para luego no tener que hacer un ternario a la hora de renderizar el siguiente componente.

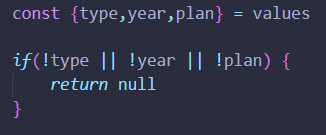


Se lo **pasamos** al componente **CarInfo.js** y guardamos toda la información que teníamos.

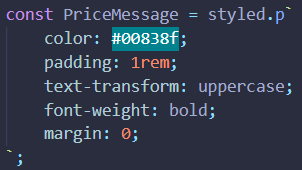
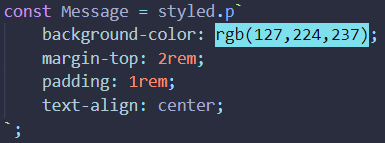


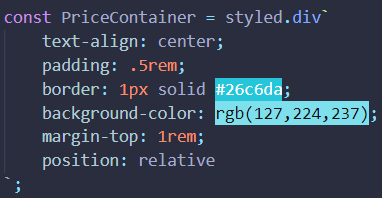
**17**. **Mostrando los datos**: creamos un nuevo componente llamado **Insurance.js,** le hacemos un style-component y le pasamos como props los datos.

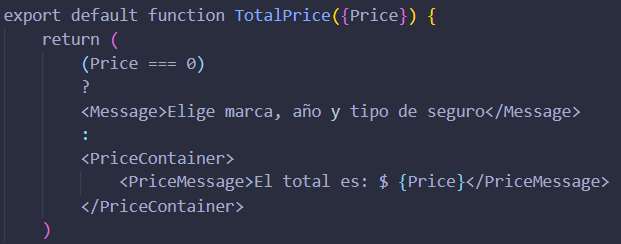
Dentro de **App.js** descomprimimos values del estado y se lo pasamos a **Insurance.js**

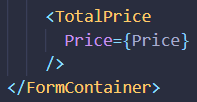
**18**. **Renderizando un componente por vez**: para evitar que la información del seguro se imprima antes de completar los datos, debemos hacer lo siguiente. Dentro de **Insurance.js** **preguntamos** si **alguno** de los **valores es nulo** y en caso de que si **retornamos como null** todo el componente. Esto es una alternativa a usar ternarios dentro de App.js

**19**. **Mostrando el monto total**: creamos el componente **TotalPrice.js,** allí vamos a hacer 3 styled-components: Message (para el mensaje que aparece antes de llenar los campos), PriceMessage (el valor total de la cotización) y PriceContainer (que va a contener al precio)



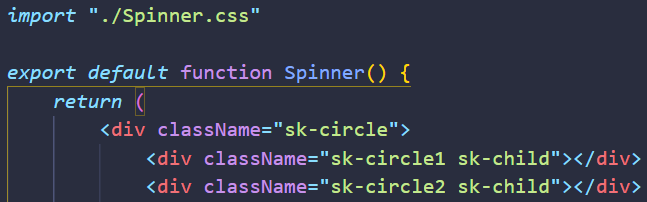
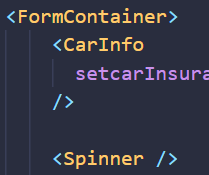


Al componente se la va a **pasar el precio total**, y lo que vamos a hacer es lo siguiente: si el **precio es igual a 0** (es el valor sin completar los campos) **queremos que salga un mensaje**, en cambio **si ya tiene un valor le queremos dar el valor total**.

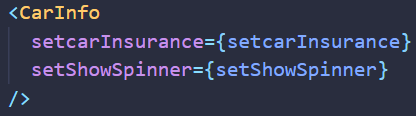
Dentro de **App.js**, **descomprimimos Price** y se lo pasamos al componente creado.

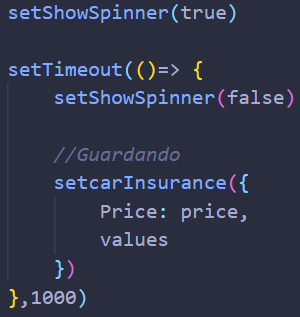


**20**. **Agregando un spinner**: vamos a entrar a [esta página](https://tobiasahlin.com/spinkit/) y seleccionamos una animación que nos guste. De allí **copiamos** el código **HTML** y creamos un nuevo **componente** llamado **Spinner.js**, a su vez creamos una **hoja de estilos** **única** para ese archivo **Spinner.css**. **Importamos** el componente dentro de **App.js** y lo ponemos debajo del formulario.



**21. Que el spinner se renderize en el momento indicado:** creamos un **nuevo estado** llamado **ShowSpinner** y lo **iniciamos** en **false**. Le **pasamos** la variable de escritura a **CarInfo.js** y luego de **completar los datos lo pasamos a true**. Luego dentro de **un timeOut** de 1 segundo **pasamos nuevamente a false** el estado y **guardamos los datos**.





Finalmente dentro de **App.js** hacemos un **ternario** **preguntando** si **showSpinner** está en **true**, en caso de que si **mostramos el spinner** y en ca**so de que no renderizamos el detalle y el precio final.**

