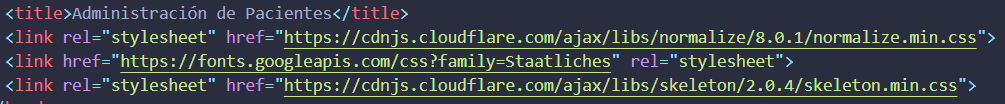
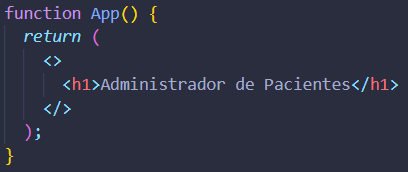
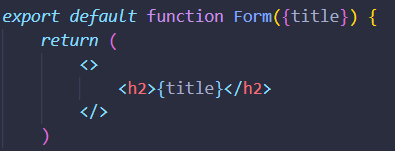
**Proyecto 2: administrador de pacientes**

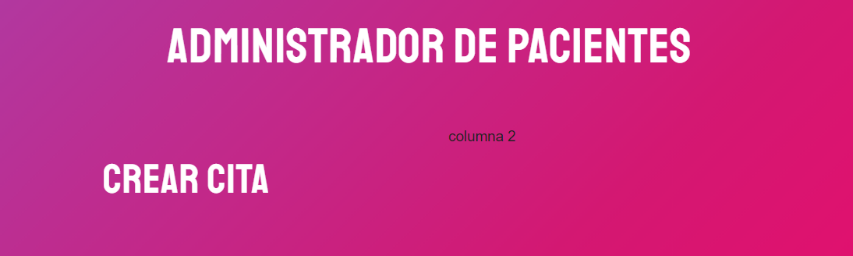
Resultado final con y sin citas cargadas:

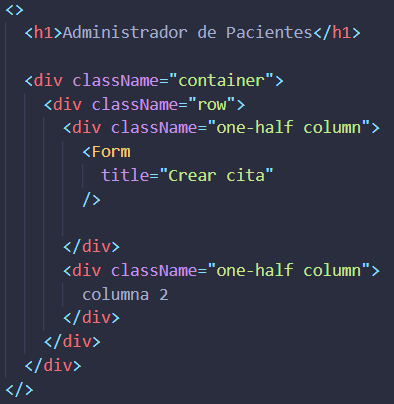
**1.** En el archivo **index.html** (dentro de public) debemos configurar dentro del **<head>** las hojas de estilos y las tipografías de Google (fines esteticos)

**2**. **Incluimos el código CSS** (que nos pasan por Github) **en el archivo index.css**. El mismo ya se encuentra enlazado dentro de index.js (src).

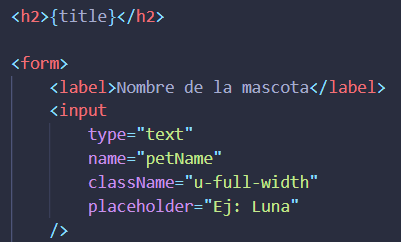
**3**. Ponemos el título del proyecto en un <h1> **dentro de App.js**

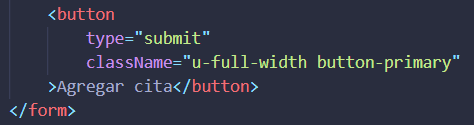
**4.** Creamos el **componente** “**Form**” que va contener a **todos** los **campos** del **formulario**, y por el momento le dejamos únicamente un título, que será pasado por props

**5.** Dentro de **App.js** **importamos** el **componente** creado y lo colocamos en la primera columna. Nótese que **todas esas clases de los <div> pertenecen al framework Skeleton** (es como Bootstrap)

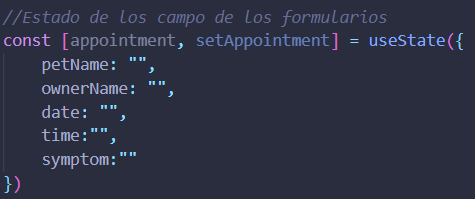


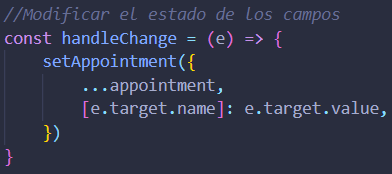
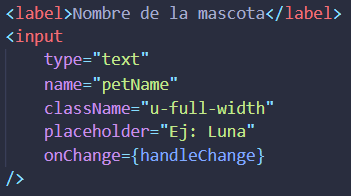
**6**. **En el componente “Form”** debemos colocar una **etiqueta** **<form>** y meter allí **todos los campos**, de forma general por el momento tendrán **4 propiedades**: **type**, **name** (nos ayudara a distinguir el campo), **className** (para que ocupe el 100% del ancho) y **placeholder** (marca de agua de ejemplo en el input)

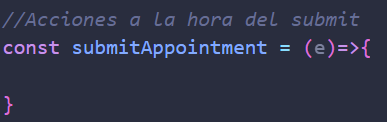
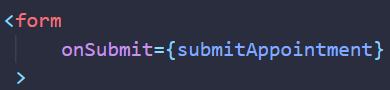


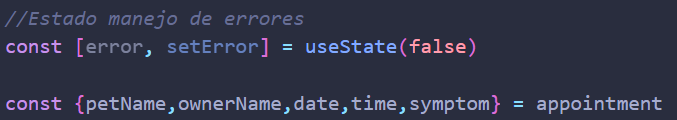


**7. En el componente “Form**” debemos requerir el **hook** “**useState**” para crearle un **estado** a los **campos del formulario**. El mismo se inicia como un **objeto** que contiene como **propiedades el “name”** de los **inputs** con el **valor** de un **string vacío**.

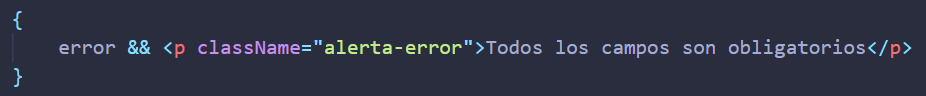


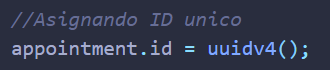
**8.** **En el componente “Form**”, **creamos una función** llamada “**handleChange**” que se va a encargar de **actualizar el objeto** con los **datos que se vayan ingresando** en los inputs. Para esto le asignamos la función creada a un evento llamado “**onChange**” que se le pone a **todos los inputs**. **Lógica de la función**: toma como parámetro el evento, llama a la variable de escritura de “**setAppointment**” **descomprime el objeto** (para que el resto de los datos no se borren) y le **cambia la propiedad del campo que está siendo modificado** (por este motivo es muy importante ponerle el “name” del input a las propiedades del objeto literal)

**9.** Ahora vamos a realizar las **acciones que se ejecutaran a la hora de darle submit** al formulario, estas son: **validar los campos, asignarle un id único a cada cita, guardar la cita** **y** finalmente **reiniciar el formulario**. Debemos **crear** una **función** y **asignársela** al **<form>** con un **evento** “**onSubmit**”.

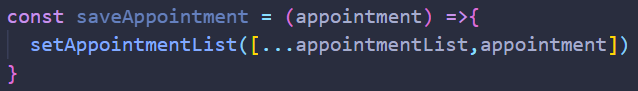
**a) Validando campos**: primero vamos a desestructurar el objeto que contiene los valores de los campos (por comodidad) y vamos a **crear** un nuevo **estado** **para los errores** que se va a **iniciar en “false”.**

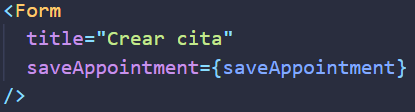
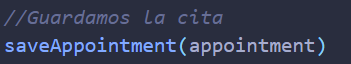
Luego dentro de la función que maneja que sucede en el Submit, hacemos un **preventDefault()** para evitar que se envié. **Preguntamos** si **alguno** de los **campos está vacío**. En caso de que alguno este vacío **cambiamos el estado de error a “true”** y retornamos la función, **de lo contrario** volvemos a poner en “**false**”.

**Finalmente** arriba del <form> ponemos un **condicional** **preguntando** si **error** está en “**true**”, y en caso de que si **imprimimos un mensaje de error.**

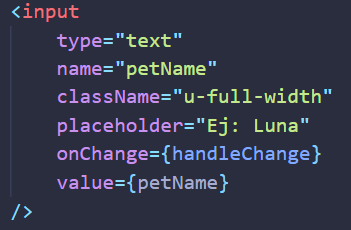
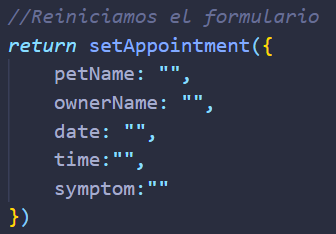
**b) Dándole ID único a cada cita**: instalamos una librería con “**npm i uuid**”, la **requerimos**. Dentro de la función le **creamos la propiedad “id” al objeto** y le **asignamos** la **función** de la librería para crear un id único. Esto nos servirá para cuando queramos mapear todas las citas y necesitemos una key única para cada una.

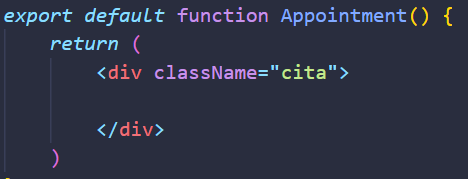
**c) Guardando la cita:** para esto debemos **crear un nuevo estado** que se va a encargar de **guardar todas las citas creadas**. Como lo vamos a usar en otro componente más adelante, este estado **lo creamos** **dentro de App.js**

Luego **creamos** una **función** que se va a encargar de **actualizar** el **estado** de la **lista de citas** **cuando se agreguen o eliminen**. Esta va a recibir como **parámetro** **una cita** y luego **modifica** el **estado** **poniendo las citas existentes y agregando la que recibió por parámetro**.

Le **pasamos** esta **funcion** al componente “**Form**”, y la **usamos** **dentro de la función que maneja el Submit** **pasándole para parámetro la cita creada**.

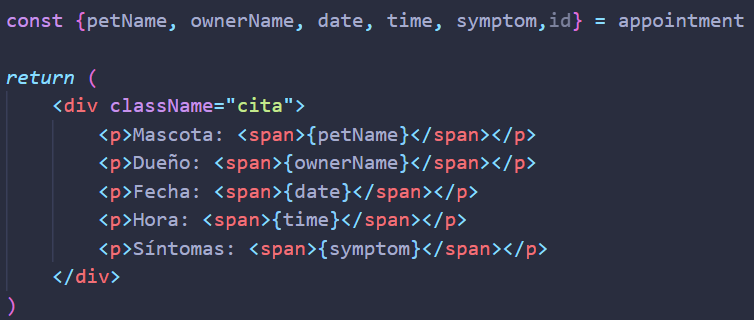
**d) Reiniciando el formulario luego de crear la cita:** para esto debemos **asignarle** a **cada input** el “**value**” de la **cita creada** y luego como **último paso de la función del Submit** **cambiar** el **estado** de los campos por strings vacíos.

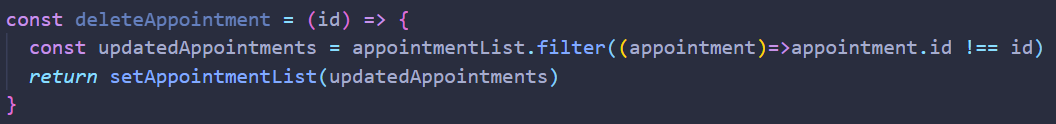


**10. Mostrando las citas creadas**: para esto tenemos que **crear** el **componente** “**Appointment**”, le ponemos un **<div>** con la **clase “cita”** para darle un estilo determinado y por el momento lo dejamos vacío.

Dentro de **App.js** **requerimos** el **componente** y hacemos un **map** de la **lista de citas**, **imprimiendo el componente en cada iteración**, pasándole como **propiedades** la key y **la cita que se está iterando**. Luego dentro del componente desestructuramos las props y las imprimimos dentro de un <p> y <span>.

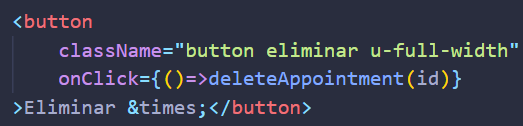


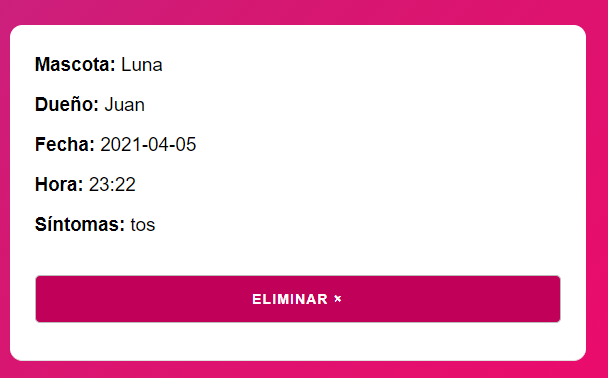


**11.** **Eliminando citas**: dentro de **App.js** debemos crear la **función** que se va a encargar de **elimina la cita deseada**. Después de crearla hay que **pasársela como propiedad** al **componente** “**Appointment**”.



Luego debemos **crear** un **<button>** en el componente y **pasarle** esa **función** en el evento “**Onclick**”.



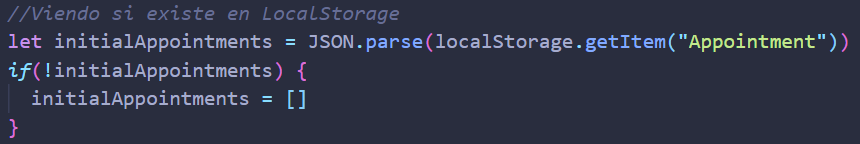


**12.** **Cambiando el titulo dependiendo si hay citas**: dentro de **App.js** debemos crear una nueva **variable** que **tenga como valor** un **if ternario** **preguntando** por el **largo de la lista de citas**. Si es **igual** a “**0**” **mostraremos un mensaje** y si es mayor otro. Finalmente ponemos la variable como título de la segunda columna.

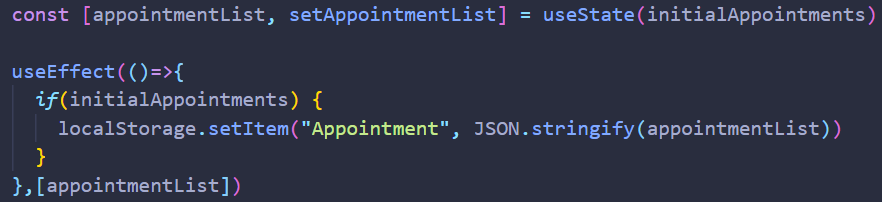




**13.** **Guardando la información de las citas en el LocalStorage**: primero debemos **definir** con que **nombre** vamos a **guardar** esa **información** dentro de **LocalStorage**. Una vez que lo tenemos, **dentro de App.js** debemos **crear** una **variable** **que se encargué de traerlo**. Para esto utilizamos **getItem**(“nombreDelItem”) y lo **pasamos** a **formato JS** con “**parse**” para que se accesible.

Luego preguntamos: **en caso de que no nos llegue nada**, le **cambiamos el valor a un array vacío.**

**14.** Importamos el **hook** “**useEffect**” y al mismo le damos la **dependencia** de “**appointmentList**” para que el **código** que se encuentra adentro, **se ejecute** cada vez que hay **algún cambio** en el **estado** de la **dependencia**.

**Preguntamos si hay citas** y en caso de que **si** las **guardamos con setItem** y la **pasamos a string (stringify).** Finalmente le **cambiamos** el **estado inicial** a “**appointmentList**” por la variable que creamos antes.

**15. Validando las props con “PropTypes”:** al final de cada componente, debemos poner qué tipo de dato es que el que se le debe pasar a cada propiedad del componente. Es meramente informativo y aparecerá un error por consola en caso de que se le pase otra cosa.

