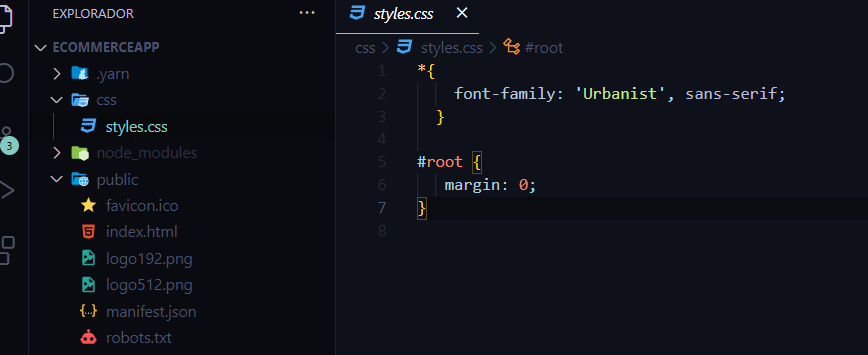
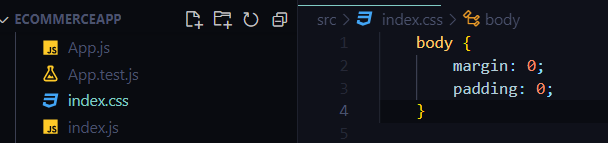
**Guía de Diseño ecommerce con React**

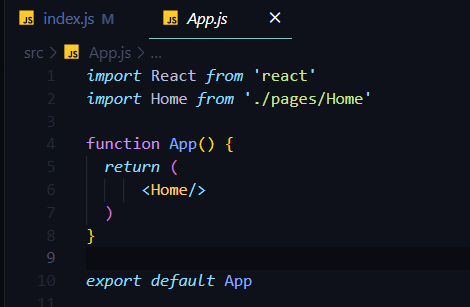
1. Se ha creado un style.css para index.html pero es más recomendable crear el estilo para el App.js, en un archivo que puede llamarse App.css. Así al ser App el componente principal se aplicará a este.

****

1. En todo caso, se recomienda indicar el tema de los márgenes y el padding para el index.js que es el punto de entrada de la aplicación. Se puede probar también por App.js. Se crea entonces el archivo index.css en caso de no existir:



1. Es recomendable dejar el index.js como viene de manera predeterminada, llamando desde dicho archivo al componente principal App.js. Por otra parte, en App.js llamamos al componente Home:



1. Este componente Home lo hemos creado dentro de un directorio de src al que llamamos pages:

Pantalla de un celular con letras

Descripción generada automáticamente

Home tiene el siguiente código:

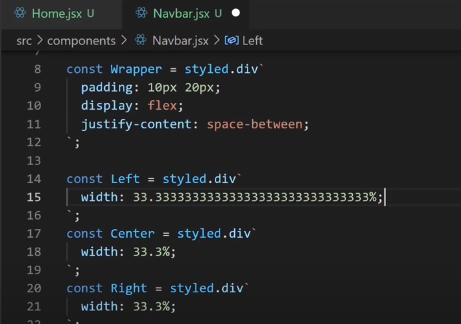
Captura de pantalla de computadora

Descripción generada automáticamente

1. En el proyecto se usó la librería Styled Component. Si usa npm puedo instalar con **npm install styled-components** o con yarn se usa **yarn add styled-components.** Luego se usa en el componente donde se vaya a utilizar importándolo:

**import styled from 'styled-components';**

1. Cuando se estaba creando el Navbar y se quería dar una distribución adecuada a los contenidos de los div “Left”, “Center” y “Right”, se intentó con:



No obstante, esto no es lo suficientemente exacto. Por eso, se implementó algo mejor: flex: 1

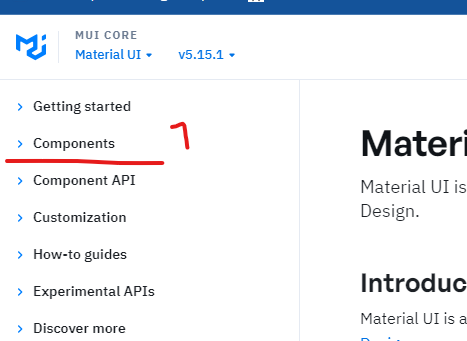
Texto

Descripción generada automáticamente

Al usar 1, decimos que cada elemento ocupará 1 unidad al dividirse el espacio en “medidas” iguales.

**Uso de íconos de Material UI**

Estos íconos en conjunto se llaman Material Icons. Los buscamos en la página de Material UI: <https://mui.com/material-ui/getting-started/> por components/ Data Display:



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Y se siguen los pasos dados allí:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para implementar algún ícono, la página nos da todo el código. Ejemplo:

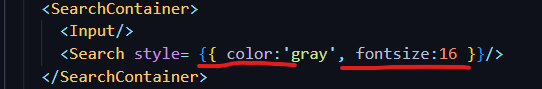
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Los estilos de Material Icons son predeterminados. Sin embargo, podemos hacer modificaciones a dichos estilos, interpolando con doble llave un estilo css dentro de cada nombre de ícono. Ejemplo:**



***Dentro del Componente Slider se usan props y estilos dinámicos***

****

**Texto

Descripción generada automáticamente**

**Gracias al uso de los props, como se ve en las capturas anteriores,** se puede pasar el valor de los parámetros desde el componente. En este caso el parámetro es direction y su valor puede ser left o right. Se hace la evaluación usando la función flecha y si se cumple una u otra condición, el valor que se devolverá será 10px. Esto ocurre porque cuando se usa el operador **&&** se devuelve el último valor si los valores anteriormente evaluados son **true**. En ambos casos direction es true.

**Uso de Hooks y Eventos en React**

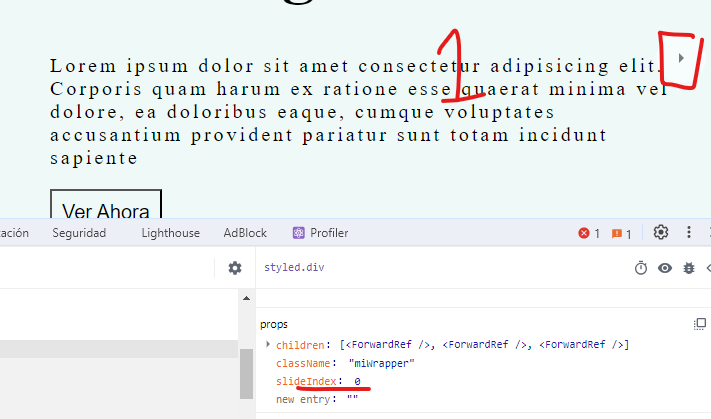
1. **Se implementaron eventos en el componente Slider:**

****

Se usó un nombre genrico de función: handClick y se pasa como parámetro bien sea “left” o “righ”, esto para la manipulación de los botones izquierdo y derecho del slider. Esto a su vez, implicará el uso de estilos css específicos que luego se irán puliendo para su funcionamiento:

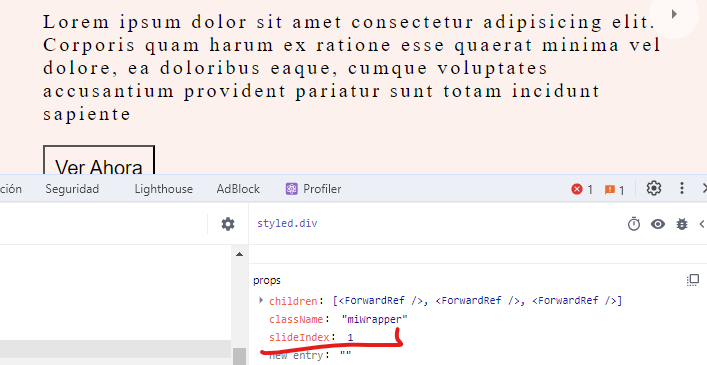
Hay que tener en cuenta que lo que queremos mover es el elemento o componente llamado Wrapper, con todo lo que tiene dentro (Image, SlideContainer, etc). Por eso debemos implementar algo que permita realizar ese movimiento. Por otra parte, todo esto va en conexión con las flechas (Arrow) izquierda (left) y derecha (right). De tal modo que al hacer clic en ellas se active el evento, haciendo llamado a la función handleClick.

Primero que todo colocamos un prop en el Wrapper, que se llame slideIndex. Temporalmente colocamos un className al div para identificarlo con React Developer Tools en el navegador. A continuación un vistazo del trabajo terminado, para entender todo el proceso. Luego se explicará cómo se llegó a este resultado final:



Esta es por así decirlo, la página número 1 del slider en conjunto. Un no movimiento del slider se lo pasamos al transform: translateX() como valor 0 (\*Nota: transform es una propiedad CSS para mover elementos en pantalla). En realidad 0 significa que no hay movimiento alguno en ninguna dirección.

Posteriormente nos encontramos esto en el siguiente slider.



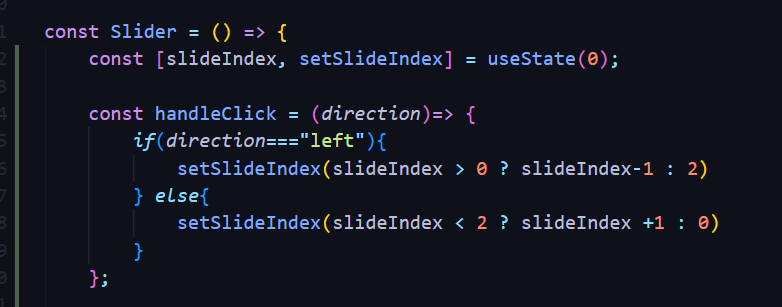
Y esto en el tercero:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Esto no significa que el valor 1 o el valor 2 por sí mismos impliquen el movimiento del slider. El truco está en multiplicar ese 1 o ese 2 o el valor que sea, por -100. Por ejemplo, si se multiplica por 1, será un valor de -100 que significa un movimiento completo de “pantalla”, en el eje indicado. En nuestro caso es el eje equis.

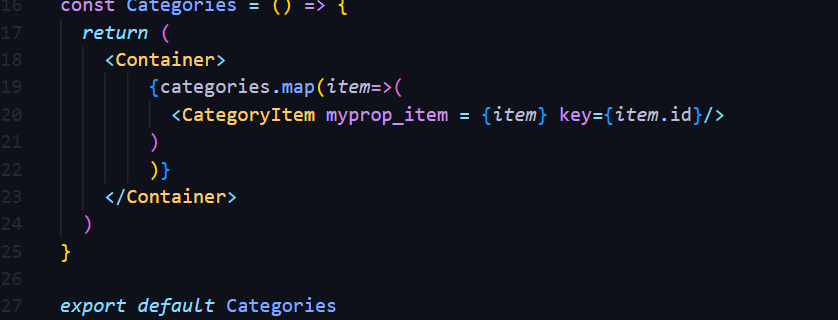
Luego, cada vez que se dé clic en el botón Arrow (flecha), sea a la izquierda o a la derecha, se aplicará la siguiente lógica:



Como se vio en capturas anteriores, slideIndex puede tomar los valores 0, 1 y 2. Y a partir de ahí se puede hacer la evaluación y el incremento o decremento de dicha variable, pero todo es gracias al hook useState(). Como ve, el valor inicial es cero. De ahí que sea useState(0). Pero la función handleclick permitirá cambiar el estado usando la otra función llamada setSlideIndex.

**Renderizado de Categorías**

Se construye el componente Categories. Para resumir, no se incluirá todo el código en este documento:

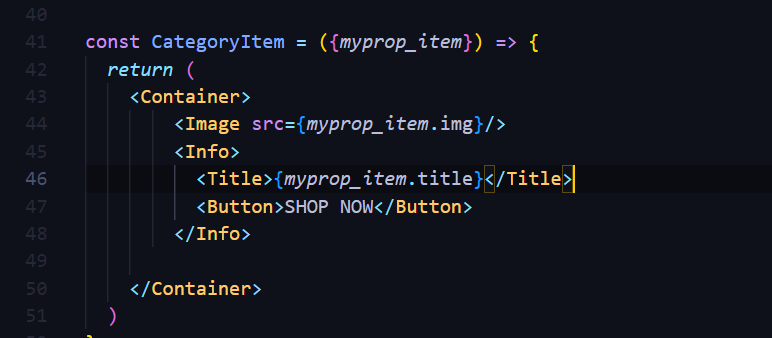


En el código anterior se usa la variable **myprop\_item** como *prop*. Esto se hace para diferenciar de la variable "item" la cual se usa para hacer el recorrido del array con el método map(). Este prop myprop\_item se pasa al componente **CategoryItem** para poder renderizar los elementos.

Categories renderiza **un único** contenedor, que a su vez renderizará **varias copias del Componente CategoryItem**, según la cantidad de registros que haya en el array categories de data.js.

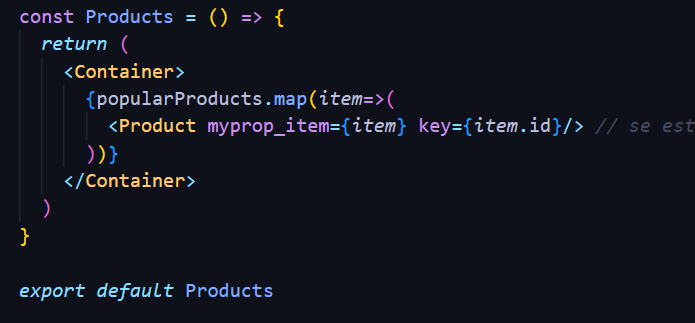
CategoryItem, renderiza solo un Contenedor base con la imagen y el nombre del producto junto con el botón “SHOP NOW” o “Comprar Ahora”. No usa como tal un recorrido de array, ya que esto se hará en el componente Categories cuando se llame a CategoryItem. Más bien, implementa el prop **myprop\_item.**

A continuación, el código de CategoryItem:



**Renderizado de Productos**

Primero creamos el componente productos, donde habrá un solo contenedor general que, a su vez, renderizará el componente Products tantas veces como ítems u objetos existan en el array respectivo. Para resumir, no se da todo el código:



En Products tenemos código de estilos como:

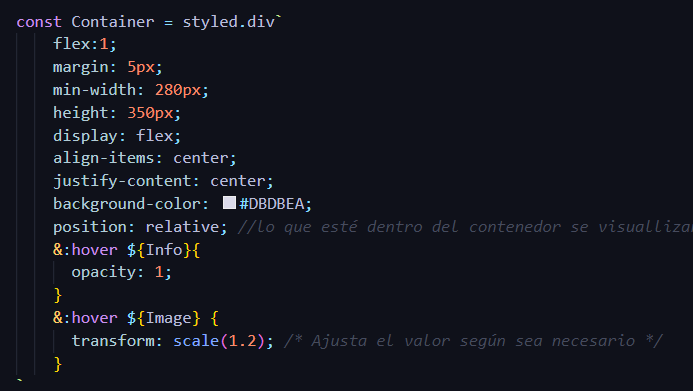
Texto

Descripción generada automáticamente

Donde opacity:0 hará invisible al elemento Info. Por otra parte, z-index: 3 le dará prioridad de visualización a Info sobre otros elementos. Su posición es absoluta con respecto al contenedor con posición relativa donde está. La transición de cualquier efecto presente en Info, será de 0.5 es decir, medio segundo. Será una transición suave (ease).

Pantalla negra con letras blancas

Descripción generada automáticamente

Tenemos también el estilo Image. Es de anotar que estos estilos no estaban al comienzo. Se habían colocado prácticamente a lo último. Sin embargo, al usar el estilo condicional en Container, no se reconocían los componentes Info e Image, ya que aparecían en las líneas de código más adelante. Fue necesario trasladar la definición de los estilos de dichos componentes, al inicio. 

Así luciría el orden finalmente:

Texto

Descripción generada automáticamente