El componente total y el cálculo

En esta sección se implementará un componente para mostrar el total de la factura. Para aquello, se van a "sumar" los ítems que se listan en **InvoiceApp.jsx**, específicamente los precios y las cantidades en la tabla **ListItemsView**. El cálculo se realizará en el servicio.

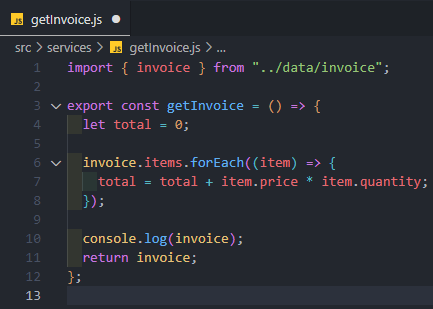
Pasos 52 al 56

Sigue estos pasos:

Paso 52: Modificar el servicio getInvoice

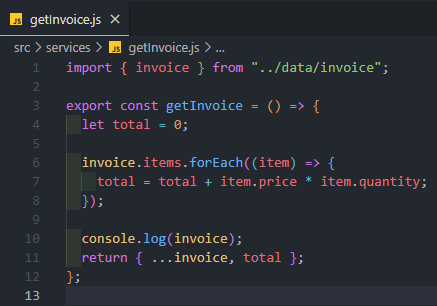
Ve al archivo **getInvoice.js** (dentro de la carpeta **services**) para modificar la función y calcular el total de la factura.

Inicializa la variable **total** y luego utiliza el método **forEach** para iterar sobre los ítems de la factura y calcular el **total**. El cálculo se realiza multiplicando el precio por la cantidad para cada ítem y se le suma al **total** acumulado.



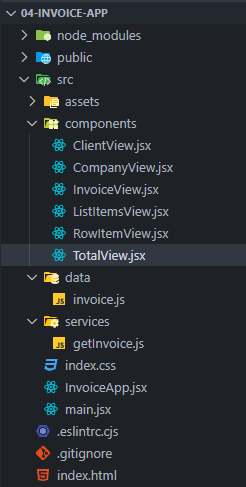
Para mantener la inmutabilidad, se retorna un objeto, en el cual va a tener una copia del objeto **invoice** utilizando el operador spread (**...**) y se le añade el total calculado.

Si en el objeto que se va a retornar se define el atributo **total** y va a tener el valor de la variable **total**, se puede simplificar simplemente colocando **total**. Esto indica que el atributo **total** del objeto tiene el valor calculado **total** (variable).



Paso 53: Crear el componente TotalView

En la carpeta **components**, crea el archivo **TotalView.jsx**. Asimismo se define una función exportable básica:

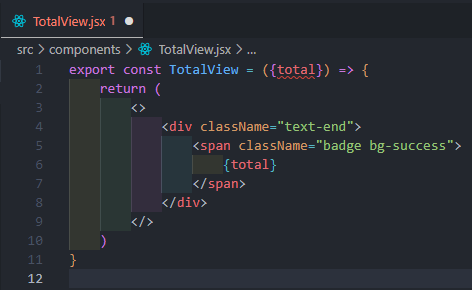




En Visual Studio Code al escribir **div.text-end** y pulsar la tecla ENTER, se crea un **<div>** con la clase de Bootstrap **text-end**, que sirve para alinear el contenido a la derecha (usando Emmet en VSCode):

Dentro del **<div>**, se crea una etiqueta **<span>** con clases de Bootstrap **badge** y **bg-success** para estilizar el **total** (abreviadamente se puede escribir **span.badge.bg-success**).

El valor del **total** se pasa como una propiedad (prop) desestructurada al componente:



Paso 54: Integrar el componente TotalView en InvoiceApp

Importa el componente **TotalView** en **InvoiceApp.jsx** y define la prop **total** sin valor por el momento.

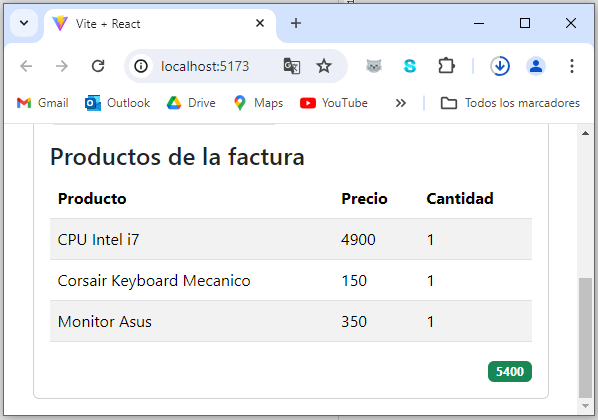
Renderiza el componente **TotalView** debajo de **<ListItemsView>**. Alternativamente, podría anidarse dentro del componente **ListItemsView**, pero esto podría acoplarlo demasiado a la tabla.



Modifica la desestructuración de la función **getInvoice** para incluir la prop **total**. Pasa el valor del atributo **total** a la prop **total** del componente **TotalView**.

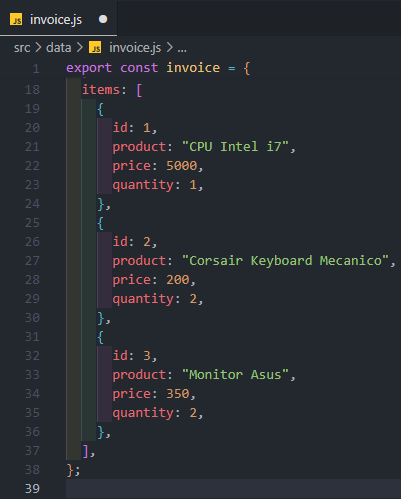


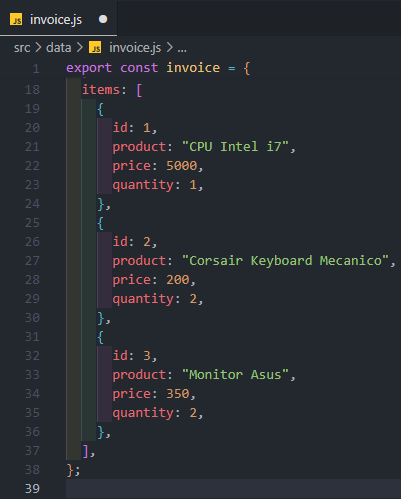
Guarda todos esos cambios y ejecuta la aplicación web. La aplicación debe mostrar el total calculado de la factura.



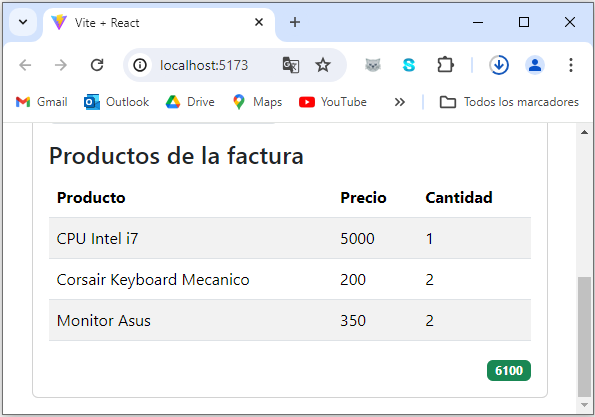
Paso 55: Probar con datos diferentes

1. Abre el archivo **invoice.js** en la carpeta **data**.
2. Modifica los valores (precio y/o cantidad) de los ítems.
3. Guarda los cambios y verifica que el valor del **total** cambie en la aplicación.





Al realizar estos pasos, se asegura que el total de la factura se calcule correctamente y se muestre en la interfaz de usuario, actualizándose automáticamente al cambiar los datos de los ítems.



Paso 56: Simplificar el cálculo del total utilizando reduce

Para simplificar el cálculo del total utilizando el método **reduce**, sigue estos pasos en el archivo **getInvoice.js**.

Se comenta todo el código antiguo que calculaba el total utilizando un bucle **forEach**.

Define una nueva constante **total** que utilizará el método **reduce** en el arreglo items del objeto **invoice**. Antes de aplicar el método **reduce**, se utiliza el método **map** para multiplicar el precio por la cantidad de cada **item**, convirtiendo así el arreglo en un arreglo de números.



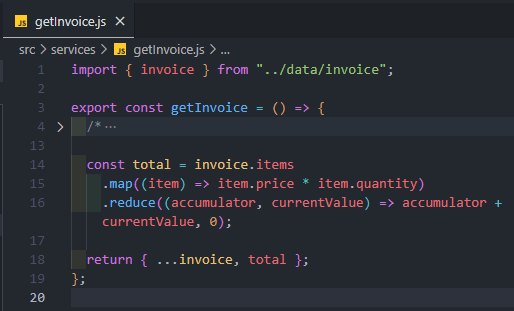
Como es una sola línea de código, se puede omitir las llaves y el **return** (porque en el método **map** está retornando el cálculo del precio con la cantidad).

El método **reduce** puede utilizarse para una variedad de tareas, como sumar todos los elementos de un array, encontrar el valor máximo o mínimo, concatenar todos los elementos de un array de cadenas, entre otros.

Acepta 2 argumentos, el primero es una función de flecha que sirve para realizar una operación con los parametros **accumulator** y **currentValue** y el segundo es el valor inicial del **accumulator**. Ten en cuenta los parámetros que se utilizan:

* **accumulator:** El valor acumulado de todas las iteraciones anteriores. Es el valor que se va acumulando a medida que avanza el proceso de reducción.
* **currentValue:** El valor del elemento actual que se está procesando.

Es por ello que se va a utilizar el método **reduce** para reducir los elementos de un array a un único valor, en este caso sumar todos los valores del nuevo arreglo obtenido mediante el método **map**. El acumulador comienza en **0** y se suma con cada valor del arreglo.



Con estos cambios, se ha logrado simplificar el cálculo del **total** al hacer el uso de métodos más modernos y eficientes como **map** y **reduce**. Ahora, el código es más limpio y legible, y la aplicación debe funcionar correctamente con el nuevo cálculo del total.

