Form, un solo state para todos los inputs

En esta sección se resumirán los estados de **productValue**, **priceValue** y **quantityValue** en un solo hook **useState**, de tal manera que se pueda manejar como un solo objeto en lugar de 3 valores separados.

Pasos 79 al 84

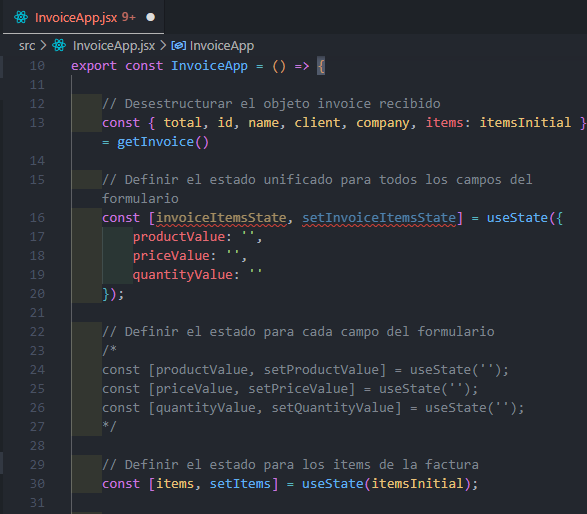
Sigue estos pasos:

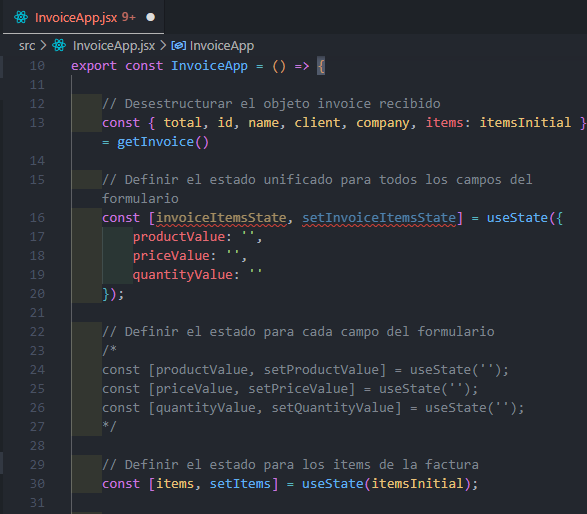
Paso 79: Agrupar los estados en un solo useState

Define un nuevo estado con el hook **useState** para agrupar los valores de **productValue**, **priceValue** y **quantityValue**; puede tener el nombre **invoiceItemsState** en lugar de **invoiceItemsValues** o **productItemsValues**. La función para modificar este estado será **setInvoiceItemsState**.

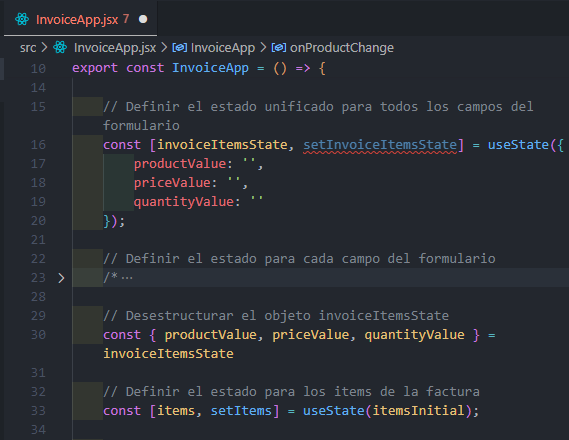
Como valor por defecto, se define un objeto con los atributos **productValue**, **priceValue** y **quantityValue**, ya que son los que se establecen en los campos del formulario (en el atributo **value** del elemento **<input>** correspondiente). Para cada uno de los atributos del objeto (similar a los valores por defecto de los estados individuales previos), se establece un string vacío como valor por defecto.

Ten en cuenta que primero se define el estado unificado y luego se eliminan o comentan los estados individuales.





Para acceder a los valores de **productValue**, **priceValue**, y **quantityValue**, se puede desestructurar el objeto **invoiceItemsState**, porque su estado contiene un objeto con los atributos mencionados.

****

Recuerda que las constantes **productValue**, **priceValue** y **quantityValue** se utilizan en la validación de los campos del formulario y en el atributo **value** del elemento **<input>** correspondiente.

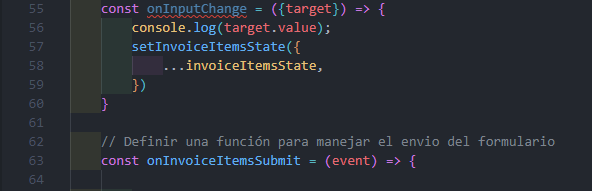
Paso 80: Crear una función unificada para manejar los cambios en los inputs

Dado que ya no se utilizan las funciones de tipo setters individuales (**setProductValue**, **setPriceValue**, **setQuantityValue**), se procede a eliminar o comentar las funciones **onProductChange**, **onPriceChange** y **onQuantityChange**. Luego, se define una función única llamada **onInputChange**, la cual se aplicará a cada uno de los inputs de manera equitativa. También se puede llamar **onInvoiceItemsChange** o **onInvoiceFormChange**.

La función **onInputChange** toma el atributo **{ target }** como parámetro, lo cual desestructura el objeto **event** para obtener el atributo **target** directamente. El **target** representa el elemento del formulario que desencadenó el evento, en este caso, uno de los elementos **<input>**.

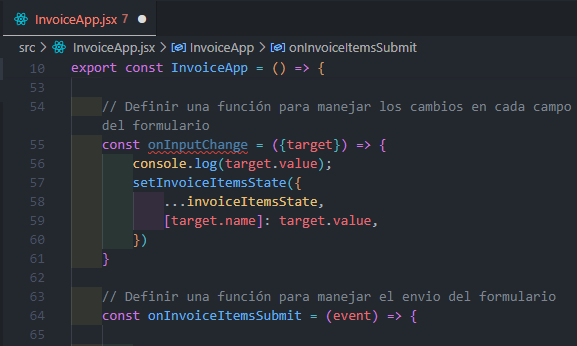
Dentro del cuerpo de la función **setInvoiceItemsState**, se utiliza el operador spread (**...**) para mantener los valores actuales de **invoiceItemsState**. Esto significa que todas las propiedades (pares de atributos y valores) del estado actual (**productValue**, **priceValue**, **quantityValue**) se copian al nuevo estado.



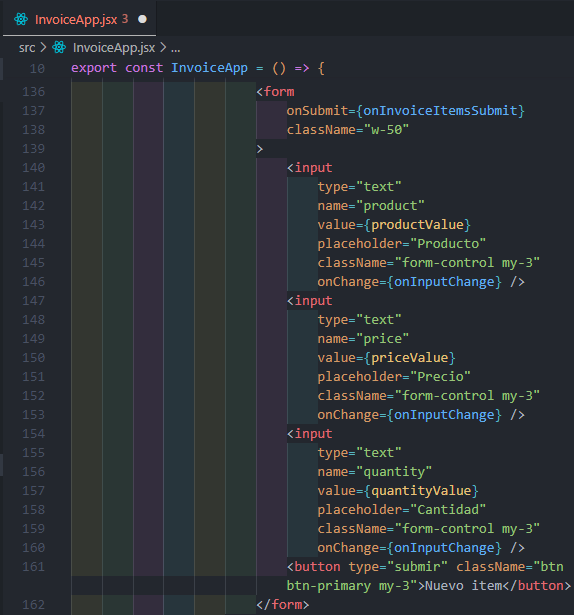


Seguidamente, se utiliza una propiedad computada del objeto para actualizar dinámicamente el campo que ha cambiado. **[target.name]: target.value** es una sintaxis que permite usar el valor de **target.name** como la clave del objeto a actualizar. Por ejemplo, si el atributo **name** de un **<input>** es **"price"**, entonces esta línea de código actualizará la propiedad **price** del estado.

Ten en cuenta que la función **setInvoiceItemsState** se llama con el nuevo objeto de estado que incluye todos los valores previos y el valor actualizado del **<input>** que cambió.



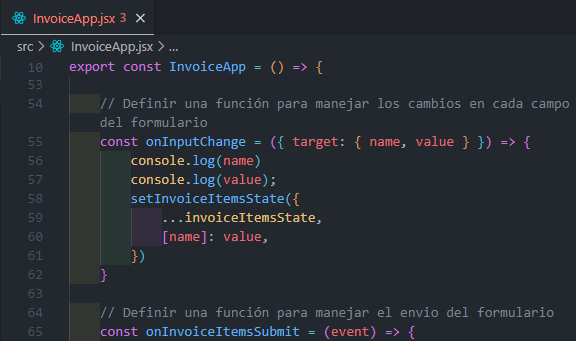
Finalmente, se asigna la función **onInputChange** al evento **onChange** de cada **<input>**. Esto garantiza que cualquier cambio en cada campo de texto será manejado por la misma función.



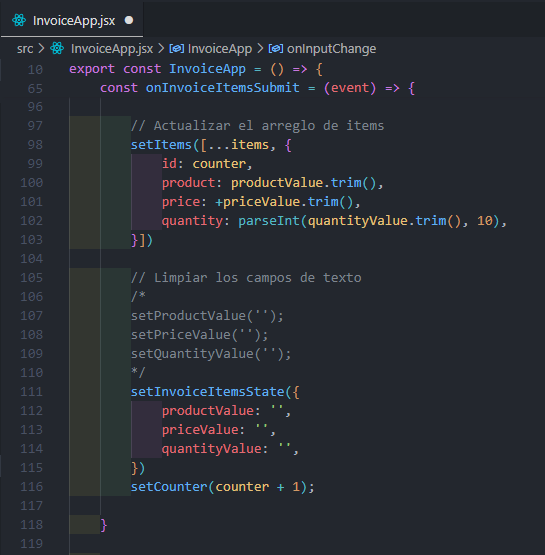
**Nota:** Esto va a llevar a un error que se va a solucionar en el **paso 83** porque la propiedad **price** del estado **setInvoiceItemsState** no existe, la propiedad **priceValue** si existe.

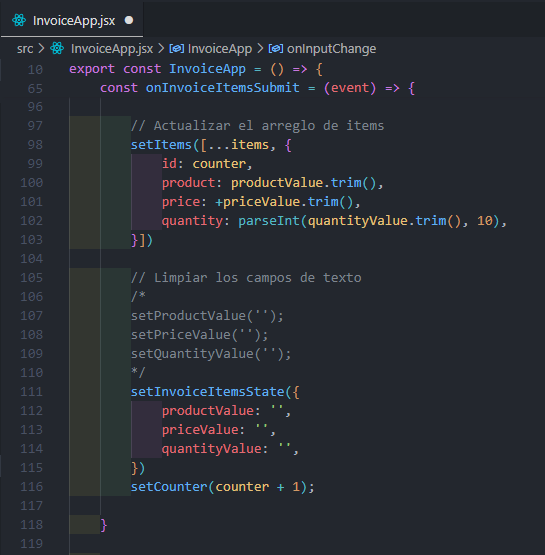
Paso 81: Desestructurar el objeto target y limpiar los campos

En la función **onInputChange**, en lugar de trabajar con la propiedad **target** completa, se pueden desestructurar directamente las propiedades **name** y **value** del **target** mediante una desestructuración anidada. Esto simplifica el código y lo hace más claro. También se añade un **console.log** para imprimir el valor de la propiedad **name**.



Por otro lado, cuando se envía el formulario, se necesita limpiar los cuadros de texto. Por lo cual, se eliminan los setters individuales (**setProductValue**, **setPriceValue** y **setQuantityValue**) y se utiliza la función **setInvoiceItemsState** para restablecer los campos del formulario a cadenas de texto vacías luego de enviar el formulario.





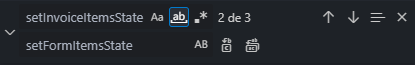
Paso 82: Reemplazar los nombres de variables

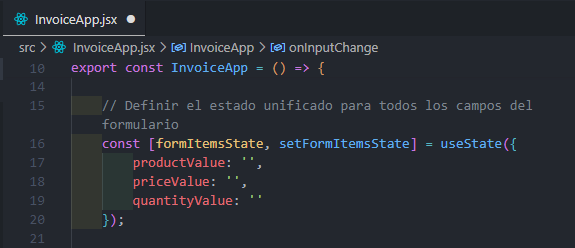
Para una mayor claridad y coherencia en el código, se procede a renombrar **invoiceItemsState** a **formItemsState**. Este cambio refleja mejor que el estado se refiere a los campos de un formulario en general, no solo a los ítems de la factura.

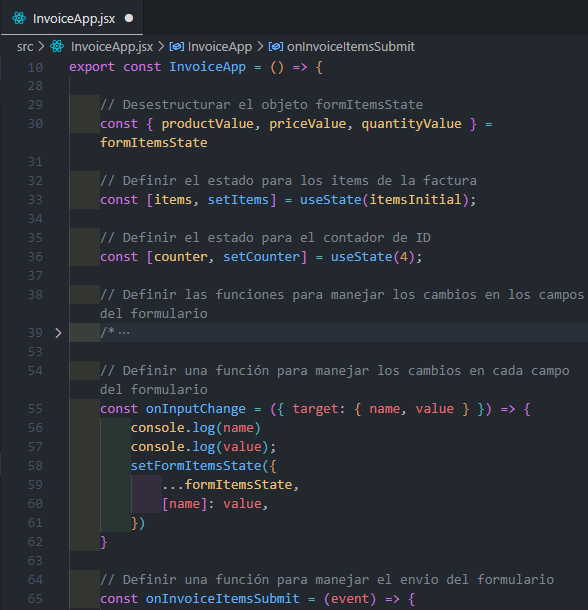
* Reemplaza **invoiceItemsState** por **formItemsState**.
* Reemplaza **setInvoiceItemsState** por **setFormItemsState**.

**Nota:** En Visual Studio Code pulsa **CTRL + H** para abrir la herramienta de búsqueda y reemplazo. Asimismo puedes marcar la opción de **Solo palabras compuestas** para hacer busquedas por palabras que coincidan exactamente.





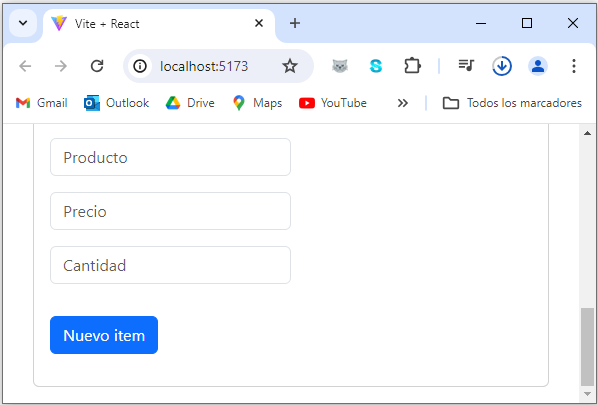




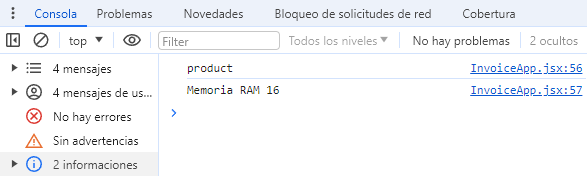


Paso 83: Corregir los nombres de campo

Durante la ejecución de la aplicación web, se puede notar que no se puede introducir texto en los campos del formulario.



Al introducir un carácter o seleccionar un producto que fue introducido previamente, se imprimira en la consola el nombre del campo del formulario (especificado en el atributo **name**) y el valor introducido (atributo **value**), esto se ha definido en la función **onInputChange**.



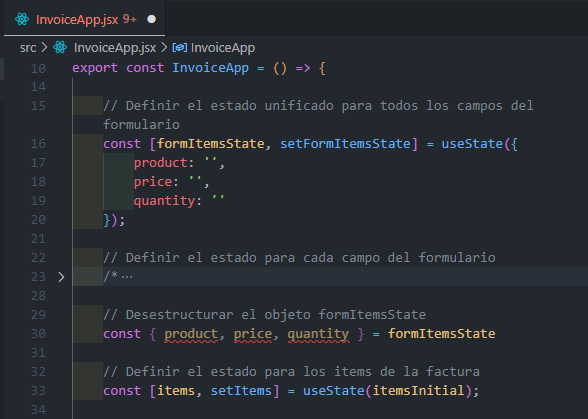
Luego de analizar el código, se ve que el error se produce porque los nombres de los campos en **formItemsState** terminan con "Value" (como **productValue**, **priceValue**, **quantityValue**), mientras que en el formulario, el atributo **name** de cada **<input>** no termina con "Value". Esto causa que la variable computada **[name]: value** en **setFormItemsState** no funcione correctamente.

Para corregir este problema, se procederá a reemplazar los siguientes terminos (puedes pulsar **CTRL + H** en Visual Studio Code):

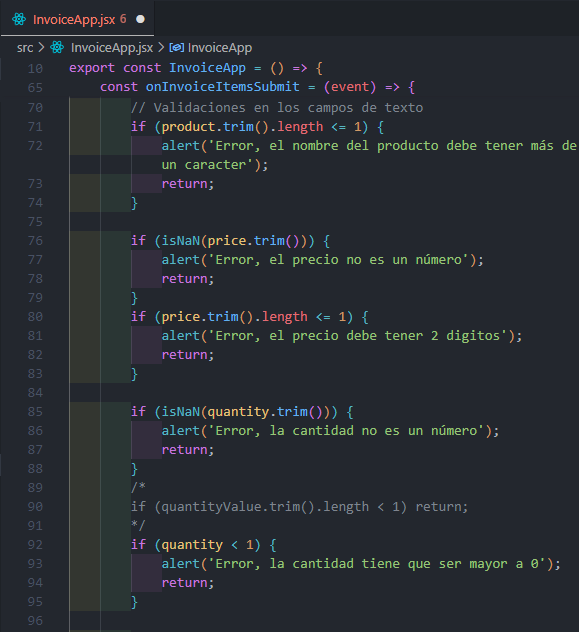
* Reemplaza **productValue** por **product**.
* Reemplaza **priceValue** por **price**.
* Reemplaza **quantityValue** por **quantity**.

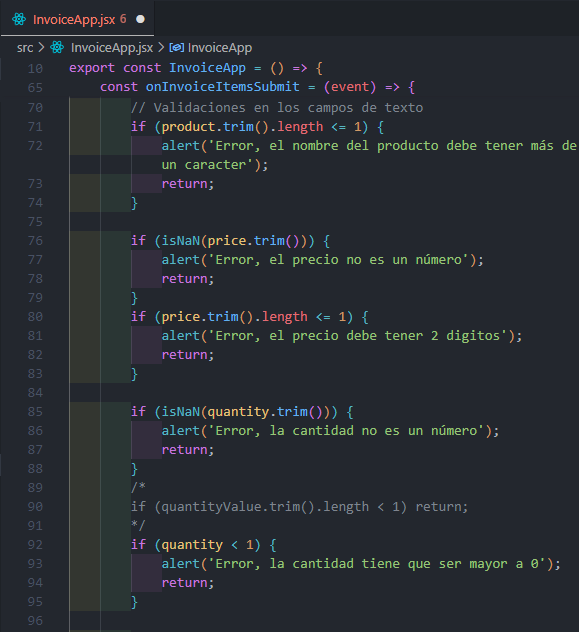
**Nota:** En Visual Studio Code, al colocar el cursor en la mitad de una palabra y presionar **CTRL + SUPR** (o **CTRL + DELETE**), se eliminará todo el contenido que se encuentra a la derecha del cursor hasta el final de la palabra.

Realiza estos cambios de manera manual primero en el estado para manipular todos los campos del formulario, específicamente en los atributos del objeto **formItemsState** y en la desestructuración del mismo.

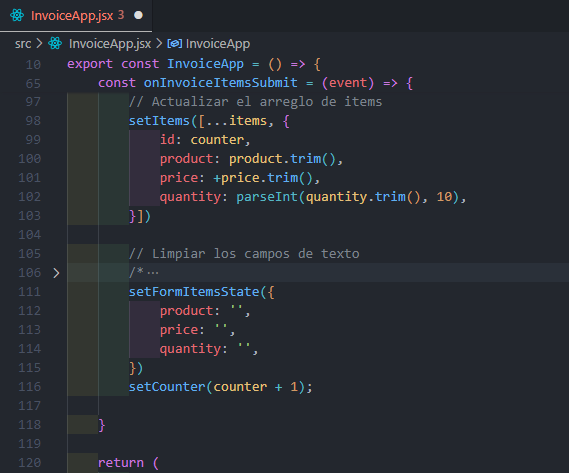


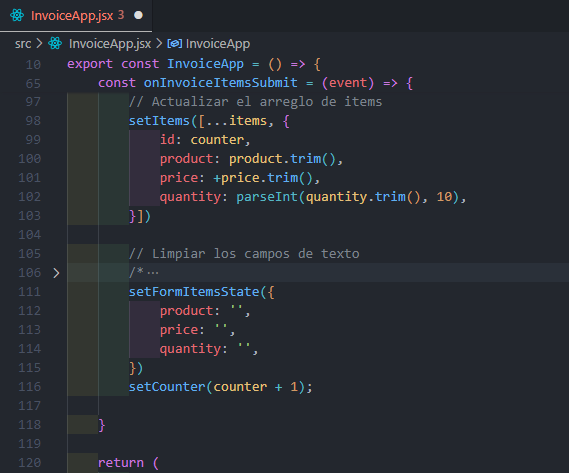
También en las validaciones de los campos de texto del formulario:



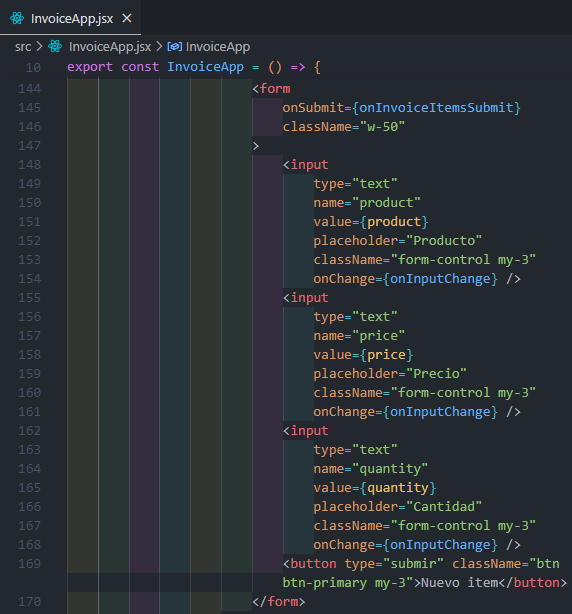


Luego en la función **setItems** (inserta los valores introducidos en los campos de texto al nuevo arreglo) y **setFormItemState** (limpia los campos de texto luego de enviar el formulario).





Y por último en los atributos **value** del formulario (el valor insertado se actualiza en el estado).



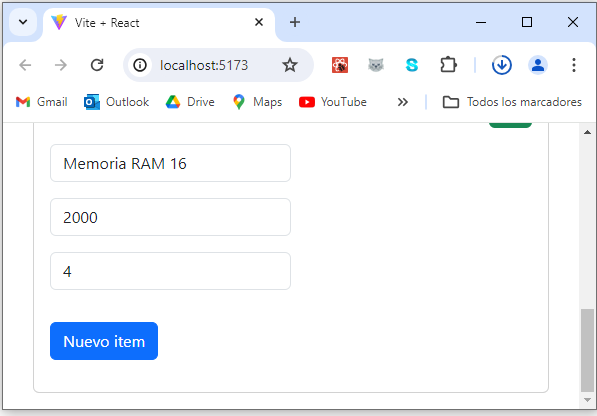
Después de realizar estos cambios, verifica que el formulario funcione correctamente y que los campos se actualicen según lo esperado.

Paso 84: Prueba en el navegador

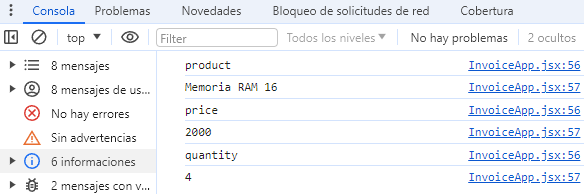
En resumen se tiene el siguiente proceso de manejo del estado del formulario:

1. Todos los campos del formulario se manejan en un solo objeto en el componente, denominado **formItemState**.
2. Se realiza la desestructuración del objeto **formItemState** para obtener los valores de cada campo.
3. Los valores de los campos del formulario, mapeados con el objeto del estado, se actualizan utilizando el atributo **value** en el HTML, manteniendo el mismo nombre que el campo en el estado.
4. Al invocar el evento **onInputChange**, se obtiene dinámicamente el nombre y el valor del campo que se ha modificado a través de los atributos del objeto **target**.
5. La función **onInputChange** actualiza el estado utilizando una variable computada para cambiar solo el campo correspondiente sin reemplazar los otros datos del estado.

Primero introduce unos datos en el formulario:

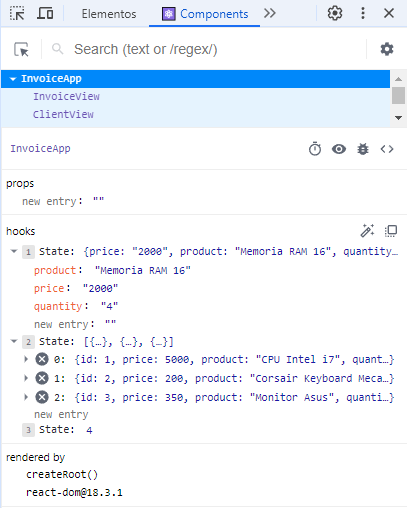


De acuerdo a la función **onInputChange**, primero se imprime el valor del atributo **name** del campo del formulario y luego el valor introducido carácter por carácter (si se selecciona uno de los valores predefinidos que aparecen al hacer clic en el campo, la consola imprime directamente el valor seleccionado)



En la sección **components** de la consola de Chrome, se selecciona el componente **InvoiceApp**.

Los hooks del componente se muestran, donde el primero contiene los campos del formulario (**product**, **price**, **quantity**), el segundo los elementos de la lista y el tercero el contador. A medida que se introducen datos en el formulario, los campos se actualizan reflejando los cambios en el estado del formulario.



Cuando se envían los datos (al pulsar el botón **Nuevo Item**), se observa que se reinician los campos del formulario. Además, cada vez que se agrega un producto, la lista se actualiza y el contador se incrementa, lo que indica que se está gestionando correctamente el estado del formulario.

