Profundizando hook useEffect

En esta sección se profundizará el uso del hook **useEffect** con varios ejemplos. Anteriormente se ha visto que cuando no se especificaba ninguna dependencia (se dejaba la lista de dependencias vacía), el efecto secundario o evento se dispara únicamente cuando se crea el componente.

Pasos 92 al 97

Sigue estos pasos:

Paso 92: Trasladar los estados al nivel superior

Es fundamental recordar que, según la sintaxis de React, los hooks **useState** deben estar al inicio del componente. Primero, corta el estado de **formItemsState** y la desestructuración del objeto, y pégalos antes del **useEffect** encargado de los datos de la factura. Es importante mantener estados separados para la factura, los ítems y el formulario (en ese orden).

A continuación, corta y pega el estado del contador (**counter**) al comienzo del componente. Dado que **itemsInitial** no se está utilizando, elimínalo de la desestructuración del objeto **invoice**. Asimismo, coloca las desestructuraciones después de obtener los estados iniciales.





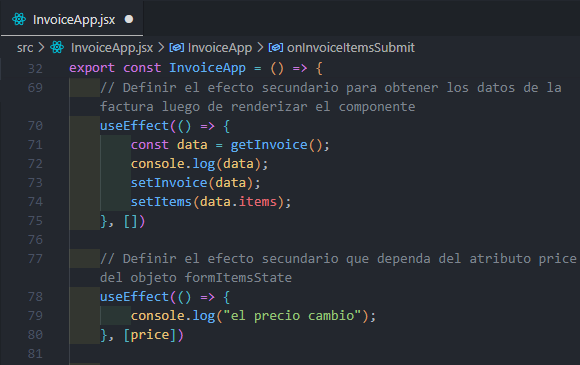




Paso 93: Crear un useEffect con dependencias

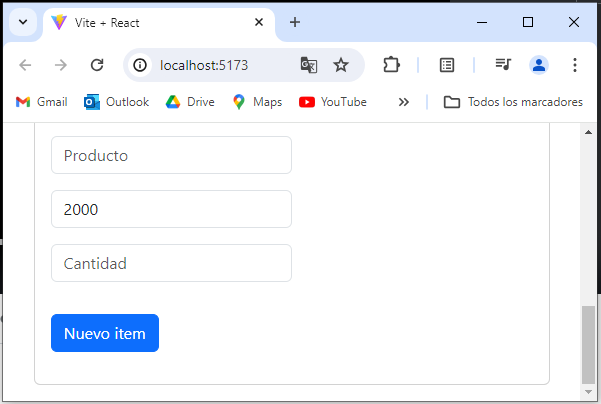
Crea otro hook **useEffect** para profundizar en su uso. Recuerda que siempre se debe definir una función de flecha sin parámetros y como segundo argumento una lista de dependencias.

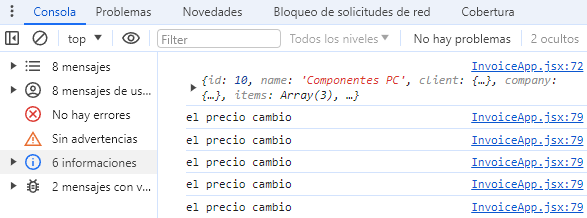
Por ejemplo, se puede utilizar el atributo **price** definido en el objeto **formItemsState** como una dependencia. Coloca **price** en la lista de dependencias del **useEffect**. Dentro de la función del **useEffect**, imprime el mensaje **"El precio cambió"** cuando el precio cambie al introducir texto en el campo del formulario **price**.



Ejecuta la aplicación web. Al recargar la página, se imprimirá el mensaje **"El precio cambió"** en la consola (esto ocurre porque el componente se monta o renderiza por primera vez).

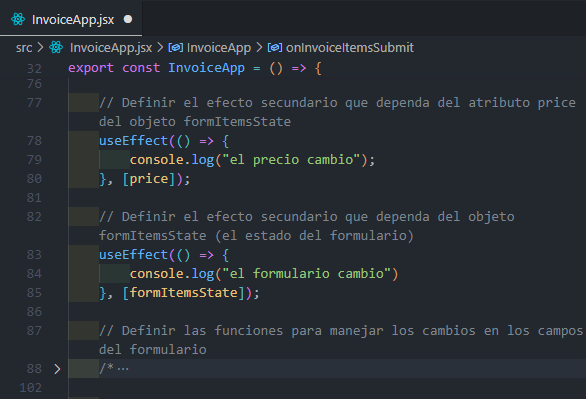
Además, al escribir texto en el campo **price**, el mensaje se imprimirá en la consola por cada carácter que se escriba, indicando que el efecto se dispara cada vez que el valor de **price** cambia. Este comportamiento es específico para el campo **price**.



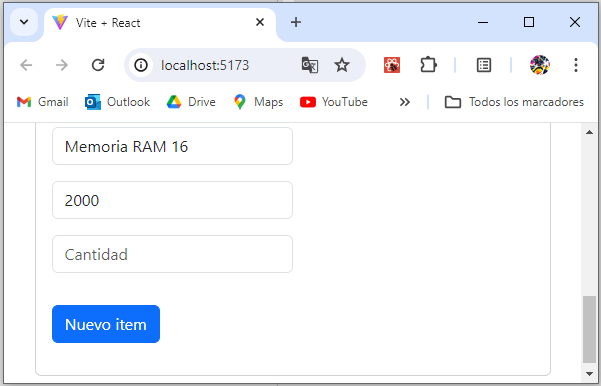


Paso 94: Crear un useEffect que dependa de todo el formulario

Para crear otro **useEffect** que dependa de cualquier campo del formulario, se necesita definir una función de flecha sin parámetros como primer argumento y como segundo argumento la dependencia **formItemsState**. Esta dependencia abarca cualquier cambio realizado en los campos del formulario, ya sea en **product**, **price** o **quantity**.



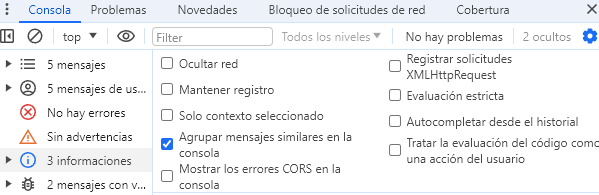
Al ejecutar la aplicación en el navegador y escribir texto en el campo **price** del formulario, se dispararán ambos efectos: el que depende específicamente del campo **price** y el que depende del **formItemsState** en su totalidad. Esto se debe a que el **formItemsState** incluye todos los campos del formulario, por lo que cualquier cambio en cualquiera de estos campos activará el efecto.





Es importante tener en cuenta que, debido a la introducción del texto carácter por carácter, el mensaje se imprimirá varias veces en la consola.

**Nota:** Para mejorar la legibilidad de la consola, en el navegador puedes agrupar los mensajes similares. Puedes hacer esto seleccionando el icono de configuración en la consola y marcando la opción **Agrupar mensajes similares en la consola**.



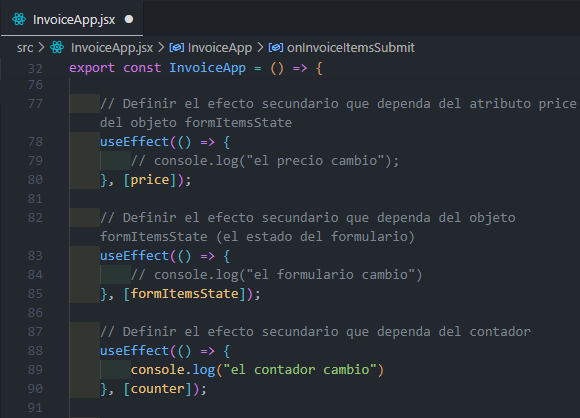
Una de las grandes ventajas de React es su capacidad para personalizar eventos basados en cualquier estado o dato de la página. Por ejemplo, se puede personalizar los eventos del contador u otros estados de manera similar a como se ha realizado con el formulario.

Paso 95: Crear un useEffect para el contador

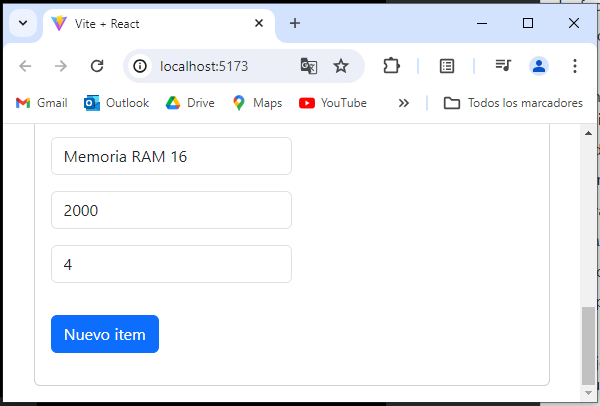
Procede a crear un **useEffect** que se dispare cada vez que cambie el contador (**counter**). Esto va a permitir ejecutar acciones adicionales cuando el contador se modifique. Para lograr esto crea una función de flecha sin parámetros como primer argumento y como segundo argumento la dependencia **counter**.

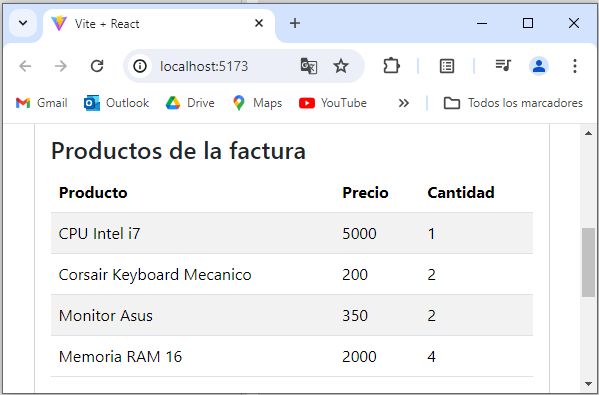
Ten en cuenta que un hook **useEffect** puede tener varias dependencias separadas por comas en el arreglo, en el cual el efecto se dispara cuando una de las dependencias haya cambiado.

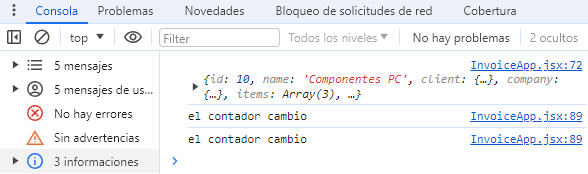
Para que no aparezcan los mensajes previos en la consola se procede a comentar los mensajes que se imprimen al disparar los efectos que dependen del campo price del formulario (**price**) y de cualquier campo del formulario (**formItemsState**)



Al actualizar la página, en la consola se imprimirá **"El contador cambió"**. Luego, al rellenar los campos de texto del formulario y crear un nuevo producto, el contador cambiará.





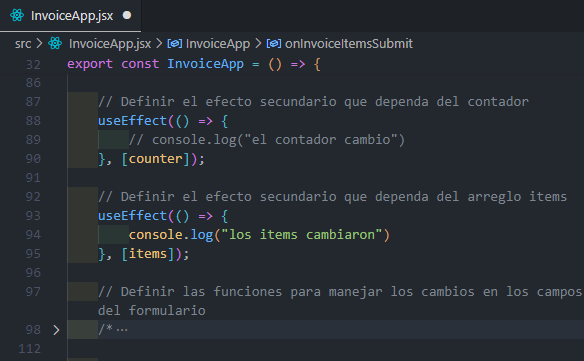


Esto se maneja en la función **onInvoiceItemsSubmit**, específicamente en el **setCounter**, que se activa cuando se crea un nuevo ítem de factura. De esta manera, el efecto se disparará cada vez que se cree un nuevo producto.

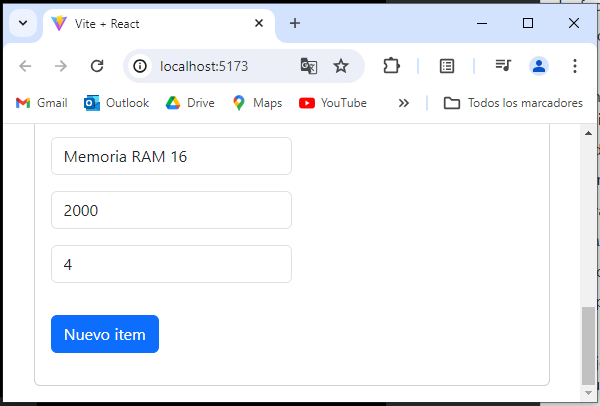


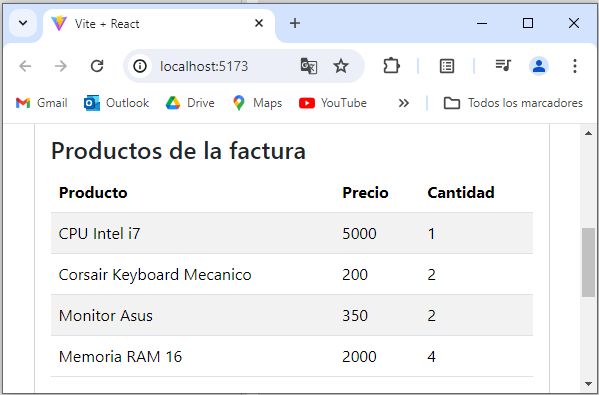
Paso 96: Crear un useEffect para los ítems

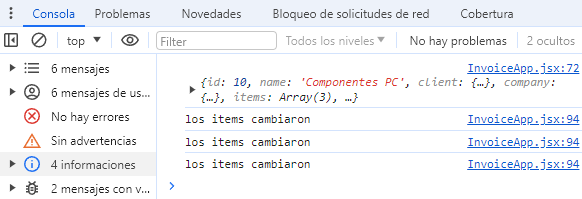
Comenta el mensaje definido en el **useEffect** del contador y crea otro **useEffect** para los ítems. Se utiliza el estado **items** como dependencia.



Cuando se crea un nuevo ítem con **setItems**, se dispara el evento ya que cambia el estado de los ítems (**items**).

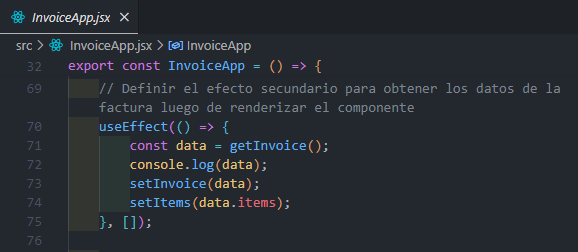




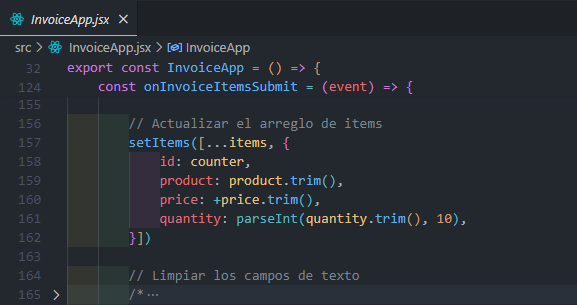


Es importante destacar que inicialmente en la consola se imprime dos veces el mensaje **"Los items cambiaron"** debido a que hay dos llamadas a **setItems** en el código fuente.

La primera llamada se encuentra dentro del **useEffect** para obtener los datos de la factura luego de renderizar el componente.



La segunda llamada ocurre cuando se actualiza el arreglo de ítems al crear un nuevo producto en la función **onInvoiceItemsSubmit**.

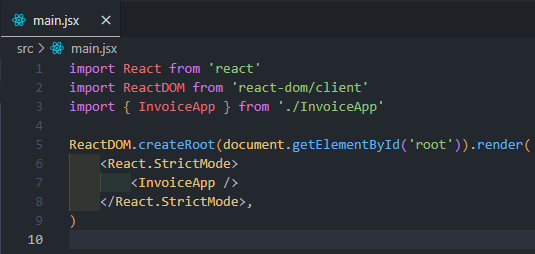


El uso de **useEffect** junto con **useState** permite manejar eventos cuando cambia el estado en combinación con la modificación de los componentes. No solamente se dispara cuando se carga el componente, sino también cuando cambian ciertos elementos que se manejan con **useState**, como elementos, valores u objetos.

En conclusión, **useEffect** es una herramienta poderosa para manejar la lógica de efectos secundarios en los componentes de React, ya sea para cargar datos, actualizar la interfaz de usuario o realizar otras acciones en respuesta a cambios en el estado.

Paso 97: Restaurar el StrictMode en el archivo main

Para restaurar el uso de **<React.StrictMode>**, simplemente quita los comentarios que lo envuelven en el archivo **main.js**.



Recuerda que el uso de **<React.StrictMode>** ayuda a identificar y solucionar problemas potenciales en la aplicación de React, cómo detectar efectos secundarios no deseados, degradados de rendimiento y otras advertencias. Es una práctica recomendada utilizarlo durante el desarrollo para mejorar la calidad y la estabilidad de la aplicación.