El service y el hook useEffect

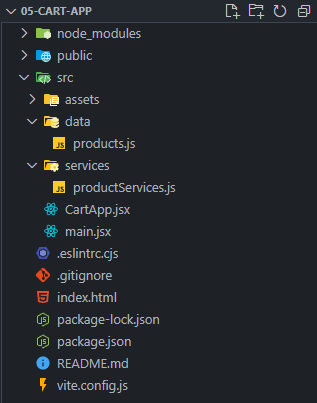
En esta sección, se creará una función para obtener los datos del catálogo de productos. Luego, se utilizará el hook **useEffect** para garantizar que estos datos se rendericen en el navegador una sola vez después de cargar la página.

Pasos 17 al 20

Sigue estos pasos

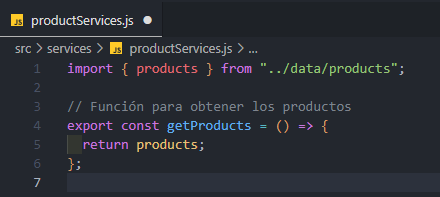
Paso 17: Crear el archivo para los servicios.

En la carpeta raíz **src**, crea una nueva carpeta llamada **services**. Dentro de la carpeta **services**, crea un archivo llamado **productServices.js**.



**Nota:** Siguiendo la convención de nomenclatura, si solo se define una función en el archivo, el nombre del archivo debe coincidir con el nombre de la función. Esta convención no es obligatoria y puede variar según el proyecto. Alternativamente, la carpeta **services** puede llamarse **helpers**.

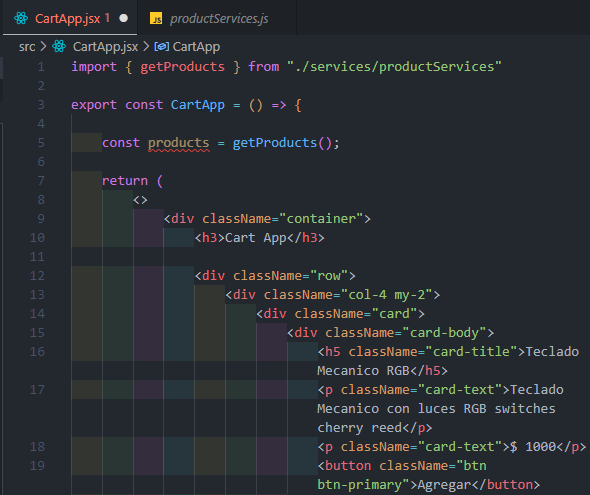
Dentro de **productServices.js**, define y exporta una función en una constante llamada **getProducts**. Esta función de flecha devolverá el arreglo de productos definido en **products.js** dentro de la carpeta **data**.



La función **getProducts** no necesita parámetros; su único propósito es devolver el arreglo de productos.

Paso 18: Integrar la función en el componente principal

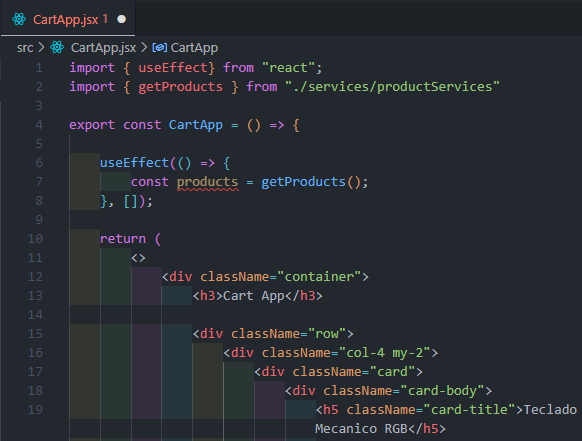
Importa la función **getProducts** desde el archivo de servicios en el componente **CartApp**, luego define una constante llamada **products** y asígnale el valor retornado por **getProducts** (un arreglo de objetos).



Este enfoque funciona bien para datos estáticos, pero si los datos provienen de forma remota, por ejemplo, desde una API o un backend, pueden surgir problemas. En particular, cada vez que se actualiza algún elemento del componente, este se vuelve a renderizar, lo que puede causar que la función **getProducts** se llame repetidamente, creando múltiples solicitudes innecesarias.

La solución a este problema es utilizar el hook **useEffect** de React. Este hook permite ejecutar código una sola vez cuando el componente se monta, evitando múltiples llamadas a la función.

Por el momento se define la constante **products** dentro de la función de flecha del hook **useEffect**, además no se agrega ninguna dependencia al arreglo, porque solamente se ejecutará al cargar la página web:



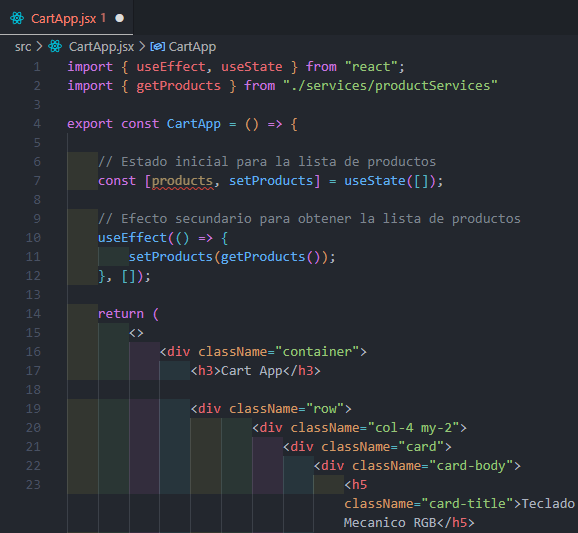
Es importante recordar que el hook **useEffect** recibe dos argumentos: una función de flecha que se ejecuta cuando el componente se monta, y un arreglo de dependencias (puede ser vacío), lo que asegura que la función solo se ejecutará una vez.

Paso 19: Definir un estado para los productos

Para manejar los productos de manera dinámica, es necesario utilizar el hook **useState** para definir el estado de los productos. Esto permite actualizar y mantener la lista de productos de manera eficiente.

Asegúrate de importar **useState** desde React, luego crea una constante llamada **products** utilizando el hook **useState**. Inicializa el estado con un arreglo vacío.

Utiliza **setProducts** dentro de **useEffect** para actualizar el estado con los productos obtenidos por la función **getProducts**, se elimina la definición de la constante **products** porque se está definiendo en el hook **useState**.

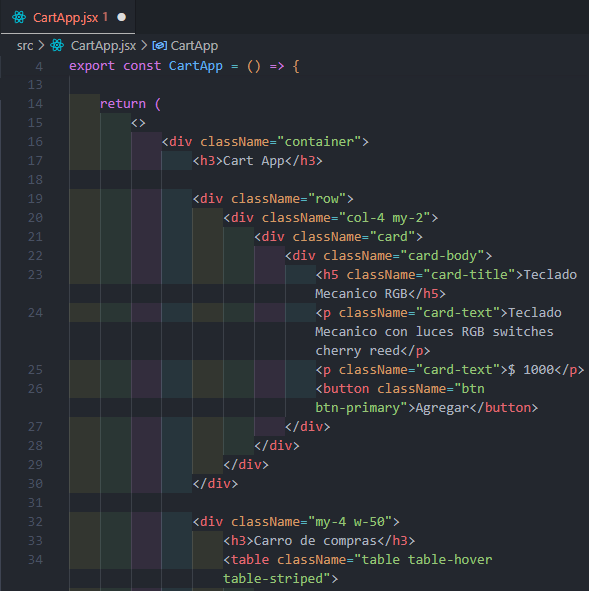


**Nota:** Recuerda importar los hooks **useEffect** y **useState** de React.

Paso 20: Reducir el código HTML e iterar con los datos de los productos

En este paso, se optimizará el código eliminando la duplicación y utilizando la función **map** para iterar sobre los productos y renderizarlos dinámicamente en el componente.

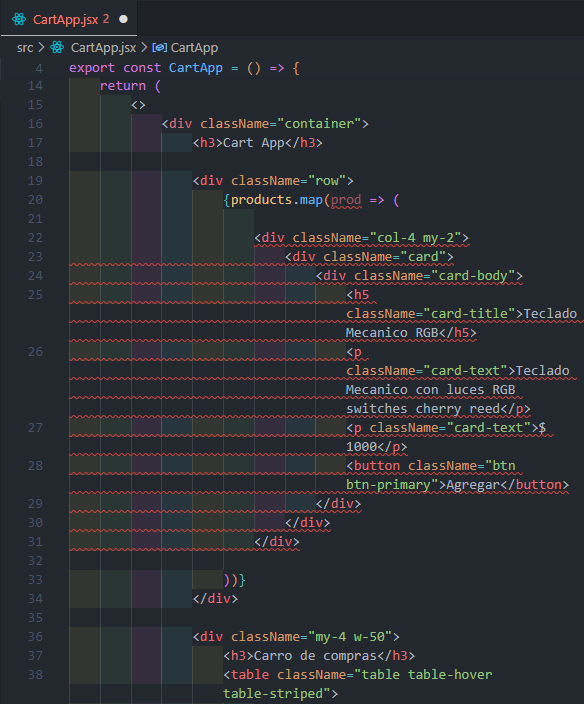
Para empezar, elimina las repeticiones de los elementos **<div>** que representan productos en el HTML (contienen las clases de Bootstrap **col-4** y **my-2**), manteniendo solo uno.



Luego se utiliza la variable **products** que está en el estado para iterar sobre la lista de productos y renderizar cada uno dinámicamente.

Se utiliza **products.map** para iterar sobre la lista de productos. Cada producto se renderiza dentro del contenedor **<div>** con la clase **col-4 my-2**.

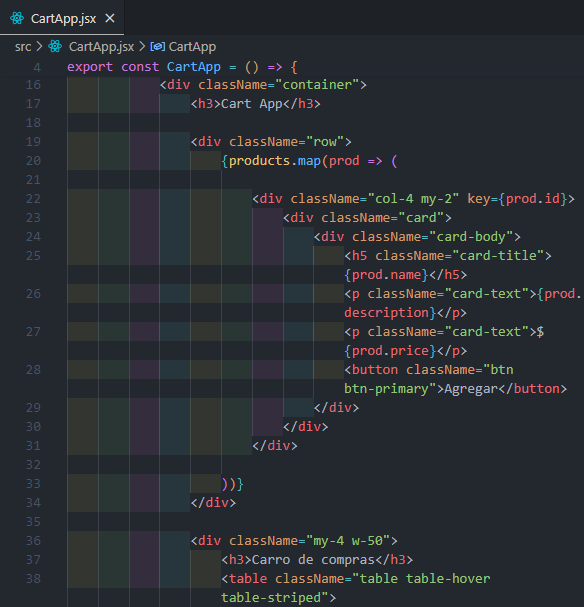
Dentro de **map**, se define una función de flecha que utilizará el parámetro **prod** para insertar dinámicamente los valores de **name**, **description** y **price** de cada producto. Por el momento se traslada el código HTML dentro del cuerpo de la función de flecha.



**Nota:** En Visual Studio Code, puedes seleccionar un bloque de código y luego usar las teclas **Alt** + **Flecha hacia arriba** o **Alt** + **Flecha hacia abajo** para mover el bloque de código seleccionado hacia arriba o hacia abajo respectivamente.

En el código HTML contenido en la función, el contenido del título reemplaza por el nombre del producto con **prod.name**, los párrafos que contienen la descripción y precio con **prod.description** y **prod.price** respectivamente.

El atributo **key** se asigna al contenedor principal de cada producto utilizando **prod.id** para garantizar que cada elemento iterado tenga una clave única, asimismo se evita el error en la consola del navegador sobre las claves únicas.



En el navegador, se muestran cada uno de los productos del arreglo de objetos definido en **products.js** con su nombre, descripción y precio respectivamente.

Además, con este enfoque, se garantiza que los productos se rendericen dinámicamente desde el estado, proporcionando una manera más eficiente y limpia de manejar la lista de productos en el componente.

