Composición de componentes

El objetivo de esta sección es dividir un componente que contiene múltiples líneas de código y etiquetas HTML en piezas más pequeñas y reutilizables, esto es una práctica común en React.

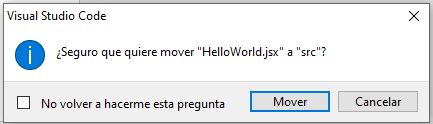
Subcomponentes en React

Podemos tener tres subcomponentes o componentes hijos en **HelloWorld.jsx** con sus respectivos props: el título (**title**), los detalles del usuario (**id**, **user.name** y **user.lastname**) y el libro (**book**). Estos subcomponentes pueden tener su propia plantilla o componente y ser reutilizados, lo cual es una ventaja fundamental de React.

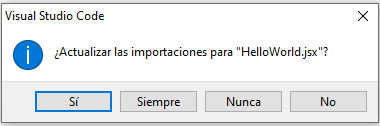
Sigue estos pasos:

Paso 1: Mover el componente

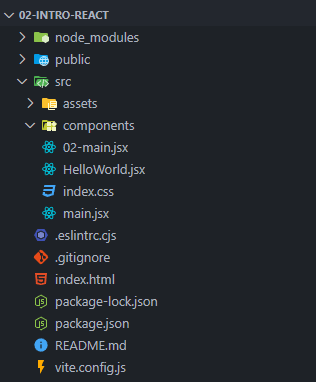
El componente **HelloWorld.jsx** puede moverse fuera de la carpeta **components** a la raíz de **src**. Esto puede hacerse simplemente arrastrando el archivo a la nueva ubicación en Visual Studio Code.

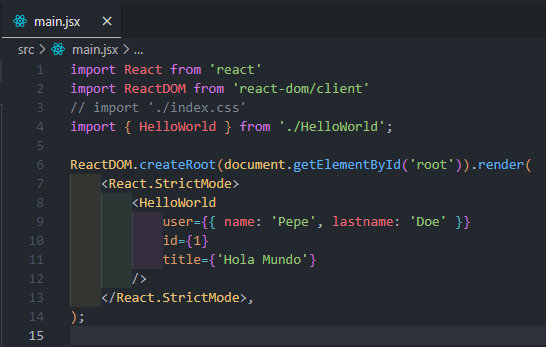


Se debe recibir una notificación para actualizar automáticamente los imports. Si no se muestra, puede ser porque no se tiene instalada la extensión **Simple React Snippets** de Visual Studio Code.



La carpeta **components** debería quedar vacía, y los **imports** en otros archivos deberían actualizarse automáticamente como en el archivo **main.jsx**.



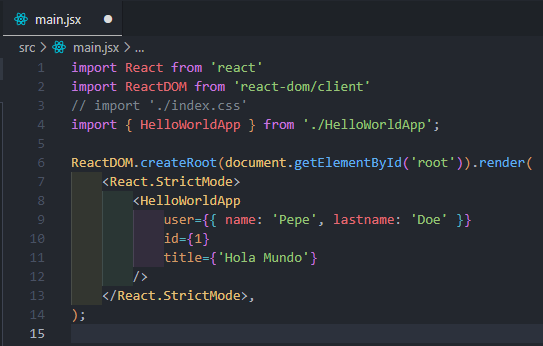


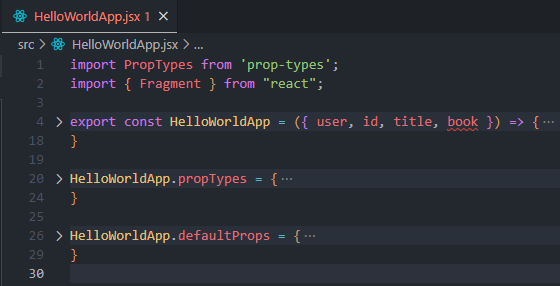
Por lo general, al importar un componente, se omite la extensión **.jsx** al final del nombre del archivo. Es importante recordar que la dirección **./** hace referencia a la carpeta **src**, el cual representa la carpeta raíz del proyecto.

Paso 2: Cambiar el nombre del componente principal y definir subcomponentes

Por convención, el componente principal de una aplicación suele llevar el sufijo **"App"** para indicar claramente su función. Por lo tanto, renombramos **HelloWorld.jsx** a **HelloWorldApp.jsx** y actualizamos los **imports** correspondientes en **main.jsx**.

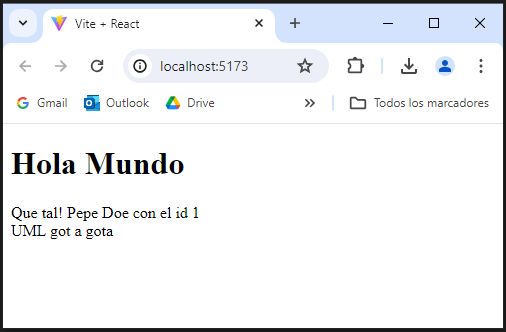
También se procede a reemplazar manualmente todo término en el código fuente que contiene **HelloWorld** por **HelloWorldApp** tanto en el componente **main.jsx** como en el componente **HelloWorldApp.jsx**. Debe quedar de la siguiente manera:



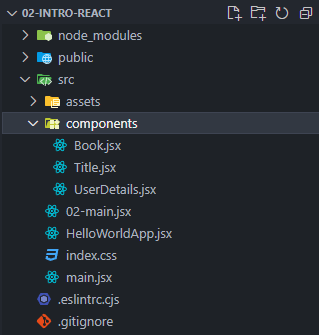


Al seguir esta convención de nomenclatura, el nombre del componente coincide con el nombre del archivo, lo que facilita la comprensión y la navegación en el código.

Al ejecutar la aplicación, no se debe mostrar ningún error en la consola y el navegador debe mostrar la siguiente pantalla:

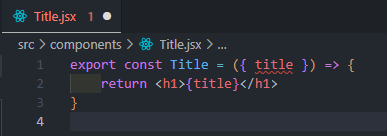


Para crear los subcomponentes de la aplicación, vamos a definir tres archivos dentro de la carpeta **components**: **Title.jsx**, **UserDetails.jsx** y **Book.jsx**. Estos archivos contendrán los subcomponentes que se utilizarán para mostrar diferentes partes de la aplicación de manera modular y reutilizable.



Paso 3: Definir el subcomponente Title

Para crear el subcomponente **Title.jsx**, comenzamos definiendo una función de flecha exportable que devuelve un elemento **<h1>** con el título proporcionado como una prop (**title**). Esto permite que el componente **Title** sea reutilizable y flexible para mostrar diferentes títulos.



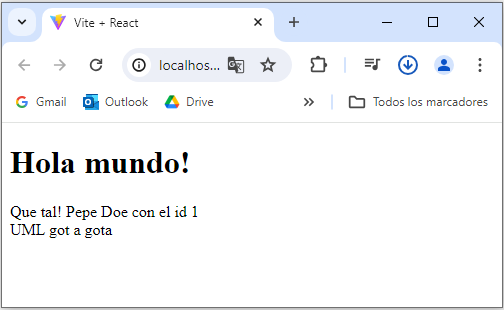
Luego, en el archivo **HelloWorldApp.jsx**, importamos el componente **Title** desde la carpeta **components** y lo utilizamos dentro del componente principal **HelloWorldApp** tal y como se muestra y le pasamos un valor para la prop **title** como hijo directo del componente **Title**.

En Visual Studio Code, automáticamente se importa el componente al escribir el nombre dentro de los corchetes (debes tener abierto ambos archivos), lo que hace que el proceso sea más rápido y eficiente.

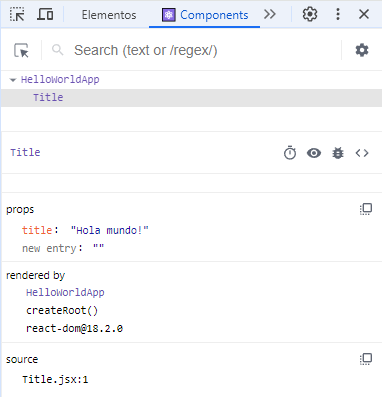




Al recargar el navegador, se debe mostrar el título **"Hola mundo!"** correctamente.

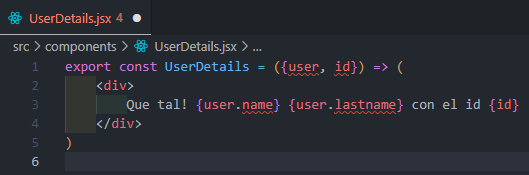


Además, en la consola de Chrome, en la sección de **componentes**, se puede ver que hay dos componentes: el componente principal **HelloWorldApp** y su hijo **Title**, que tiene la prop **title**.



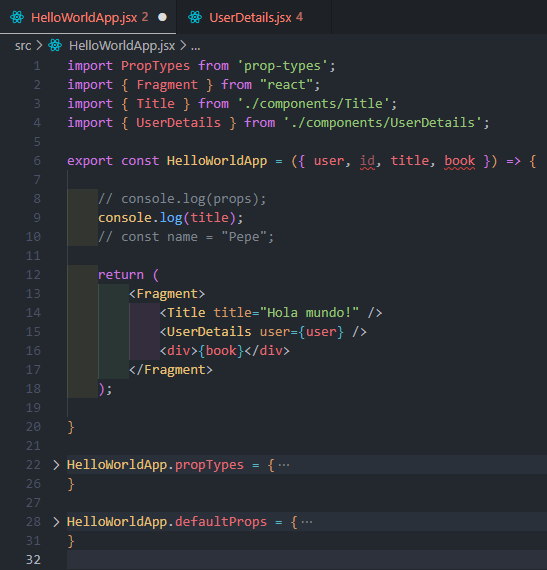
Paso 4: Definir el subcomponente UserDetails

Para simplificar la definición del subcomponente **UserDetails.jsx**, podemos utilizar una función de flecha sin el **return** y las llaves. De esta manera, las props **user** e **id** se definen y la función devuelve directamente el JSX (la plantilla HTML como un objeto).

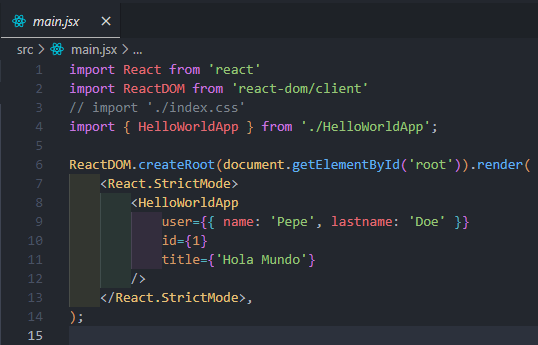


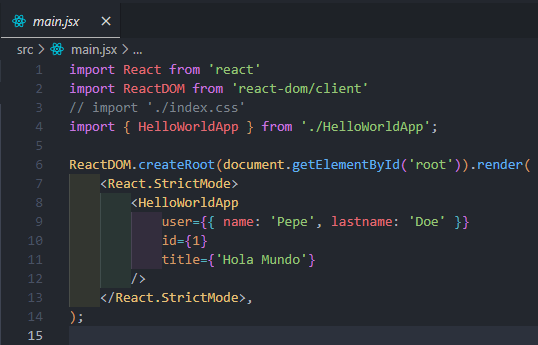
En el componente padre **HelloWorldApp.jsx**, importamos el componente **UserDetails** y lo utilizamos pasando un valor para la prop **user**, que se encuentra en la desestructuración de las props del mismo componente **HelloWorldApp**.

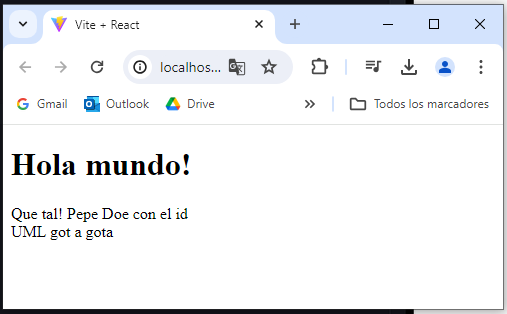
El valor para la prop **user** se recibe del componente principal **main.jsx** como un argumento de la función **HelloWorldApp**, y este componente se lo pasa a **UserDetails**.



En el archivo **main.jsx**, renderizamos el componente **HelloWorldApp** y le pasamos las props necesarias:

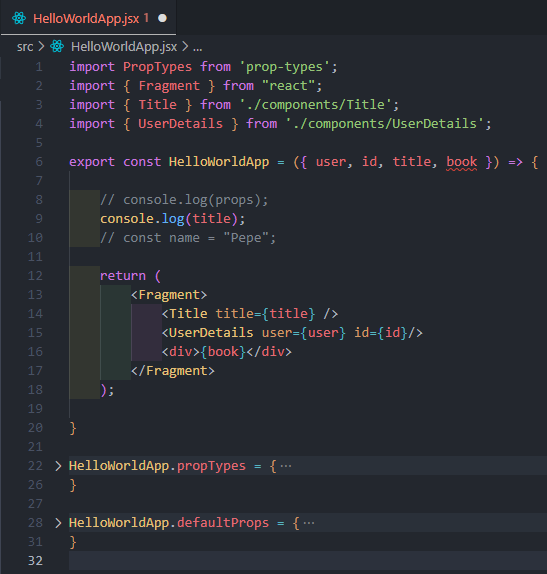


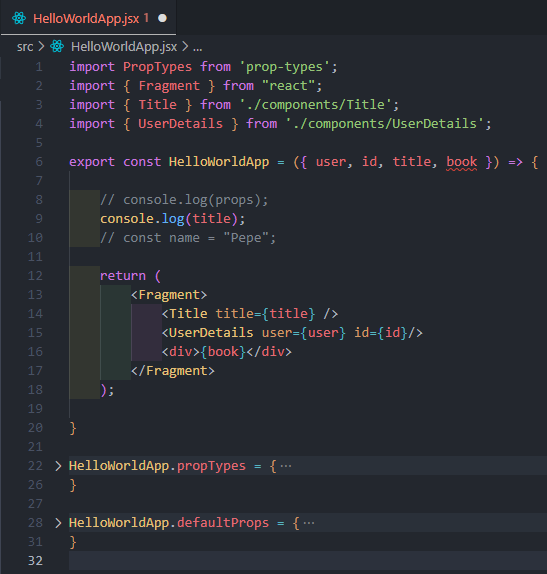


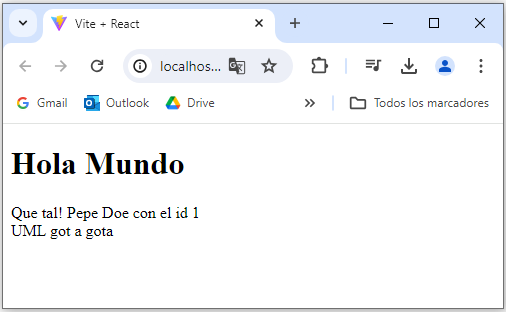


**Nota:** Podemos considerar quitar el parámetro **title** de la función **HelloWorldApp** si no es necesario recibirlo como argumento y, en su lugar, pasarlo directamente como una prop desde **main.jsx**. Todo depende de la estructura y la lógica de nuestra aplicación.

En este caso, el valor para el título (parámetro **title** definido en **<Title>**) se pasa como una prop desde el componente padre, **main.jsx**. Además, se define la prop **id** en el subcomponente **<UserDetails>**, que va a ser igual al valor definido en el componente padre **main.jsx**.

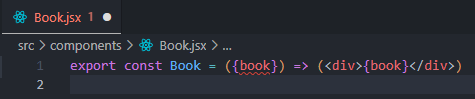




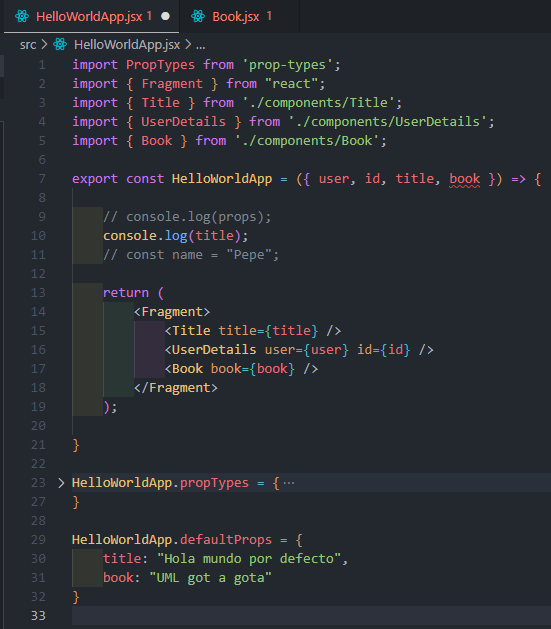


Paso 5: Definir el subcomponente Book.jsx

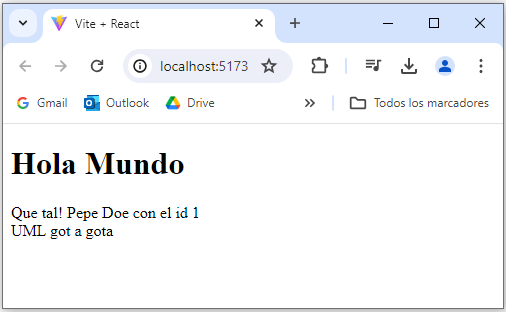
Para completar nuestra aplicación, definimos el subcomponente **Book.jsx**. Este componente simple se encarga de mostrar el título del libro. Usamos la desestructuración en las propiedades para acceder a la prop **book** directamente.



Luego, en nuestro componente principal **HelloWorldApp.jsx**, importamos el componente **Book** y lo incluimos en la renderización junto con los otros subcomponentes.



Esta división del componente principal en subcomponentes más pequeños nos permite descomponer la interfaz de usuario en partes más manejables y reutilizables.



Como se puede notar en el navegador, para la prop **book** del componente **Book** se establece el valor por defecto definido en **defaultProps**.

Con esta organización de componentes, nuestra aplicación se vuelve más modular y fácil de entender. Cada componente tiene una responsabilidad clara y puede ser reutilizado en diferentes partes de la aplicación o en otros proyectos. La composición de componentes es una práctica fundamental en React que promueve un código limpio y mantenible.