Configurando PropTypes para validaciones en las propiedades

En esta sección nos vamos a centrar en la configuración de la validación de los valores definidos en las propiedades de un componente utilizando la biblioteca PropTypes.

La biblioteca PropTypes

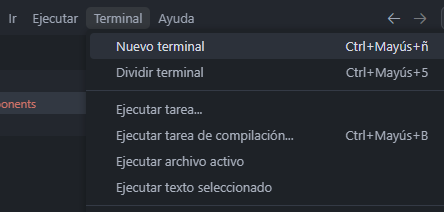
PropTypes es una herramienta invaluable para administrar propiedades en React. A diferencia de simplemente establecer valores por defecto, PropTypes proporciona una solución más sólida y desacoplada para validar las props recibidas y definir valores predeterminados cuando sea necesario.

Esta biblioteca nos permite especificar los tipos de datos esperados para nuestras propiedades, lo que ayuda a garantizar que nuestras aplicaciones funcionen de manera más predecible y resistente a errores. Además, podemos hacer que ciertas propiedades sean obligatorias, lo que mejora la robustez y claridad del código.

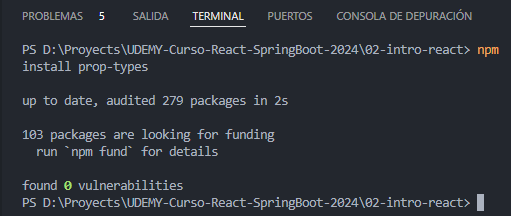
Instalar la dependencia prop-types

Para empezar a usar PropTypes, necesitamos instalar la dependencia **prop-types** en nuestro proyecto. Es importante recordar que Vite no incluye PropTypes por defecto, así que debemos añadirlo manualmente a nuestro archivo **package.json**.

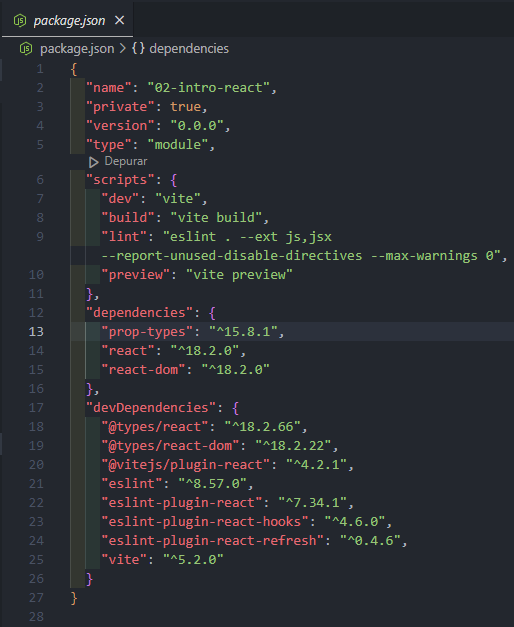
Para hacerlo, abrimos una nueva terminal en Visual Studio Code o en cualquier otra herramienta como Powershell, pero nos aseguramos de estar en la raíz del proyecto (para aquello se ejecuta el comando **cd** y la ruta hacia el proyecto; además se puede listar los archivos y carpetas mediante el comando **ls**).



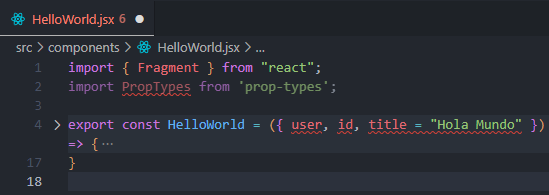
Luego, ejecutamos el comando **npm install prop-types**.



Una vez instalado, podemos cerrar la consola y verificar en el archivo **package.json** que **prop-types** esté incluido como una dependencia.



Luego, nos dirigimos al componente **HelloWorld** e importamos **PropTypes** manualmente mediante un **import** de la siguiente manera como se muestra en la imagen:

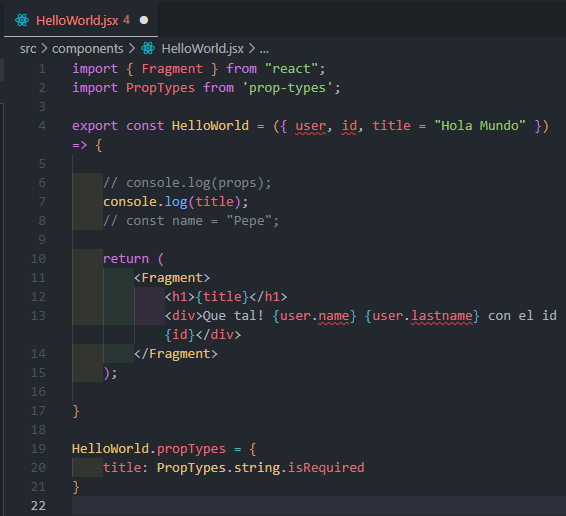


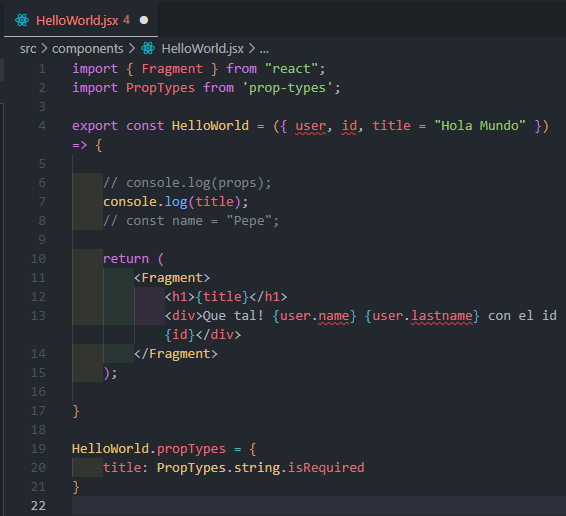
Ten en cuenta que con PropTypes, podemos establecer un contrato claro entre los componentes, mejorando la calidad y la mantenibilidad del código en nuestras aplicaciones React.

Validar el tipo de dato y hacerlo obligatorio

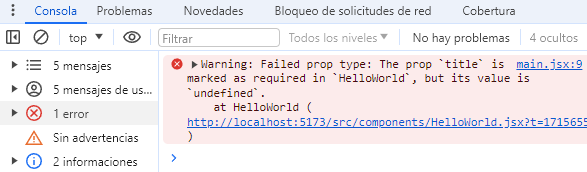
Ahora que hemos importado **PropTypes** en nuestro componente **HelloWorld**, podemos definir claramente el tipo de datos esperado para cada propiedad utilizando **PropTypes**. Además, podemos hacer que ciertas propiedades sean obligatorias utilizando la función **isRequired**.

Por ejemplo, en nuestro componente **HelloWorld**, podemos definir que la prop **title** sea de tipo string y que sea requerida utilizando **PropTypes.string.isRequired**:



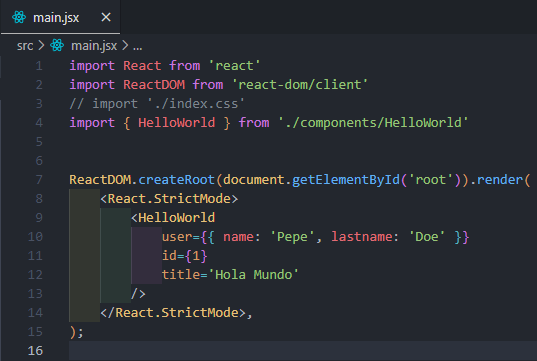


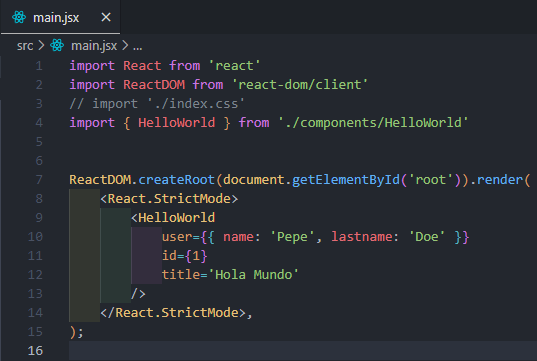
Si guardamos estos cambios y ejecutamos la aplicación, notaremos que funciona bien en la página web. Sin embargo, es posible que obtengamos un error en la consola.



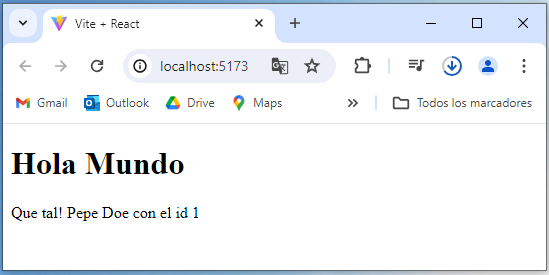
Este error indicará que la prop **title** ha fallado la validación, ya que está marcada como requerida pero se pasa como **undefined**. Esto puede ocurrir si no se proporciona la propiedad **title** en el componente principal (**main.jsx**), a pesar de que hemos establecido su valor por defecto en **HelloWorld.jsx**.

Para resolver este problema, vamos a asegurarnos de proporcionar un valor para la prop **title** en el componente principal. Por ejemplo:



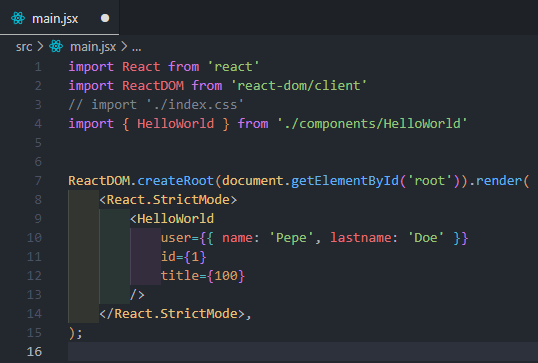


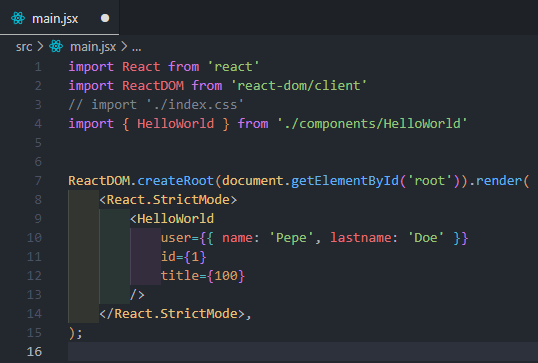
Guardamos los cambios y ahora verificamos que el error en la consola haya desaparecido. Ahora la validación de **PropTypes** se realiza correctamente y la aplicación web funciona sin problemas.

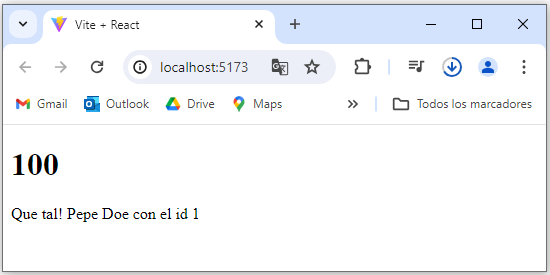


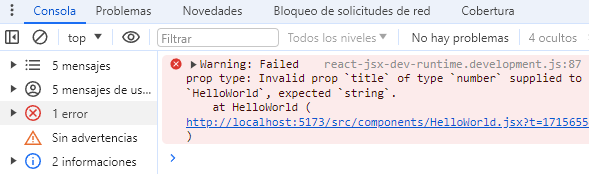
Problemas con los tipos de datos

Ahora, si intentamos pasar un título como un número en lugar de una cadena de texto, obtendremos un error en la consola a pesar de que la aplicación se visualiza correctamente en el navegador (recuerda que el valor se coloca entre llaves para validar el tipo de dato).



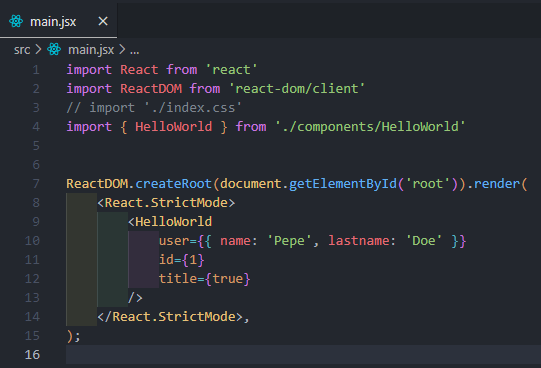


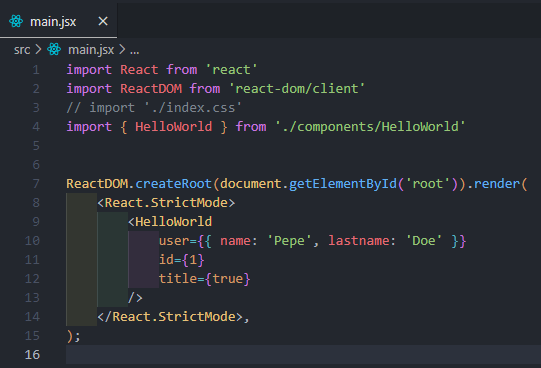


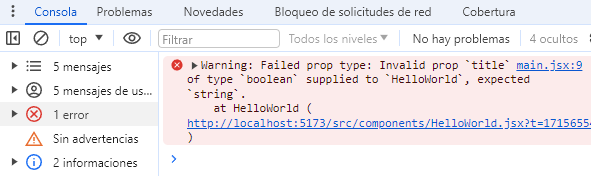


El error indicará que el tipo de dato es inválido, ya que el título es de tipo number pero se esperaba que fuera de tipo String. Esto demuestra cómo podemos ser estrictos en los tipos de datos que aceptamos y hacer que ciertas propiedades sean obligatorias.

Otro escenario común es pasar un valor booleano como título, por ejemplo, asignando el valor **true**:

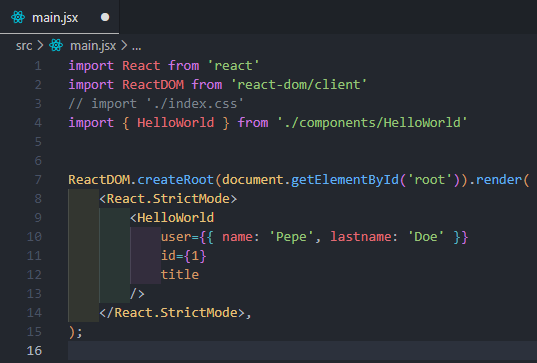


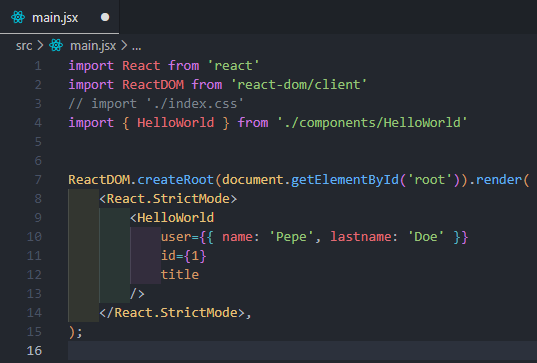




En este caso, la consola mostrará un error indicando que el tipo de dato es inválido, ya que el **prop.title** fue manejado en el componente **HelloWorld** y se esperaba una cadena de texto en lugar de un booleano.

Incluso si utilizamos un atajo como definir solamente **title** sin un valor asignado, que es equivalente a **title={true}**, obtendremos el mismo error.

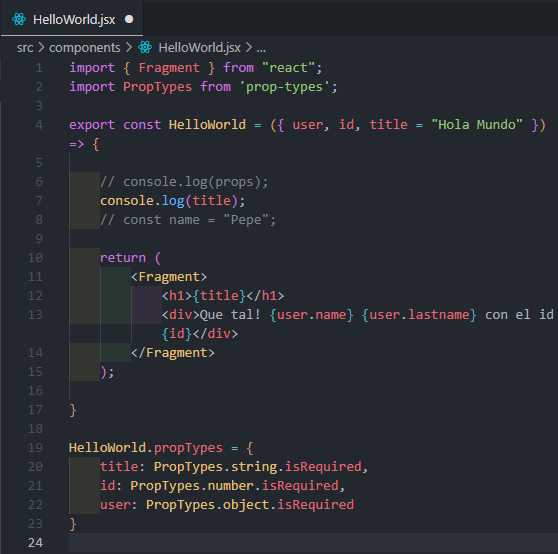




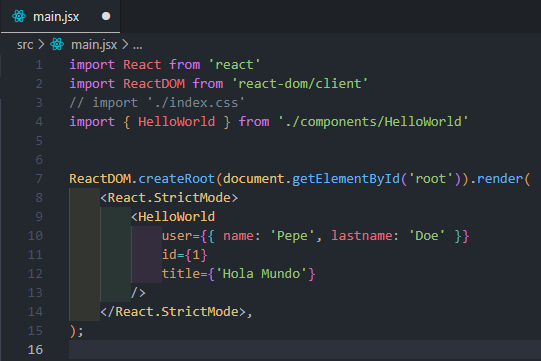
Estos errores muestran como PropTypes nos ayuda a garantizar la integridad de nuestros datos y mejorar la robustez de nuestras aplicaciones React.

Manejo de varios tipos de datos con PropTypes

Es esencial definir los PropTypes para cada propiedad en nuestros componentes React. Por ejemplo, para el **id**, debemos asegurarnos de que sea un número y, opcionalmente, podemos hacerlo requerido. Por otro lado, para el **user**, debe ser un objeto.

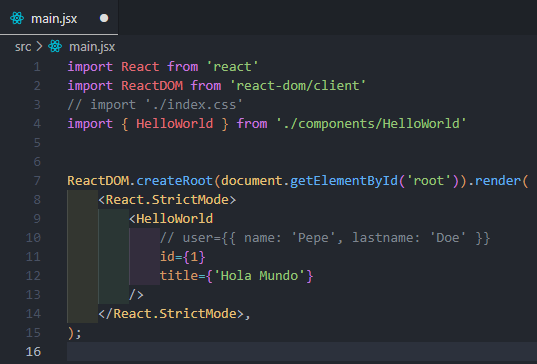


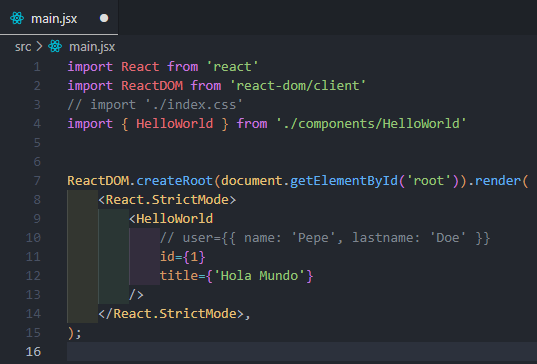
En el componente **main.jsx**, podemos ver cómo se pasan estas props (se restaura el valor de **title**):

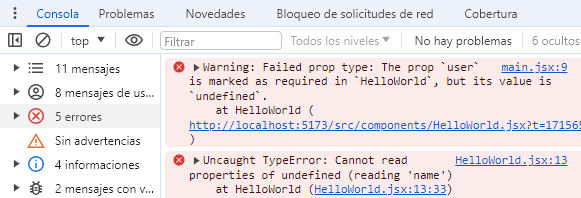


Es importante recordar que si omitimos una propiedad requerida, recibiremos un error en la consola que nos indica que propiedad falta.

Por ejemplo, si eliminamos (o comentamos) la prop **user** en **main.jsx**, recibiremos un error indicando que user es requerida en el componente **HelloWorld**, pero su valor no está definido.







Procedemos a restaurar el último cambio del archivo **main.jsx** para la siguiente sección.

