|  |  |
| --- | --- |
| C:\Users\user\Desktop\React image.jpg  Clase react | **Apuntes Curso React Desde Cero**  **Instructor: Nicolas Shurman.**  **ADSO**  **ID:2873711**  **22/09/2025**  **Centro de Procesos Industriales**  **Regional Caldas** |

Contenido

[**React** 3](#_Toc215659636)

[**Introduccion** 3](#_Toc215659637)

[**Temario** 4](#_Toc215659638)

[**Requisitos** 4](#_Toc215659639)

[**Qué es React ?** 5](#_Toc215659640)

[**Configuración de Ambiente** 9](#_Toc215659641)

[**Creando una App en React** 11](#_Toc215659642)

# **React**

### **Introduccion**

React es la biblioteca para crear interfaces de usuario de las más populares del mundo.

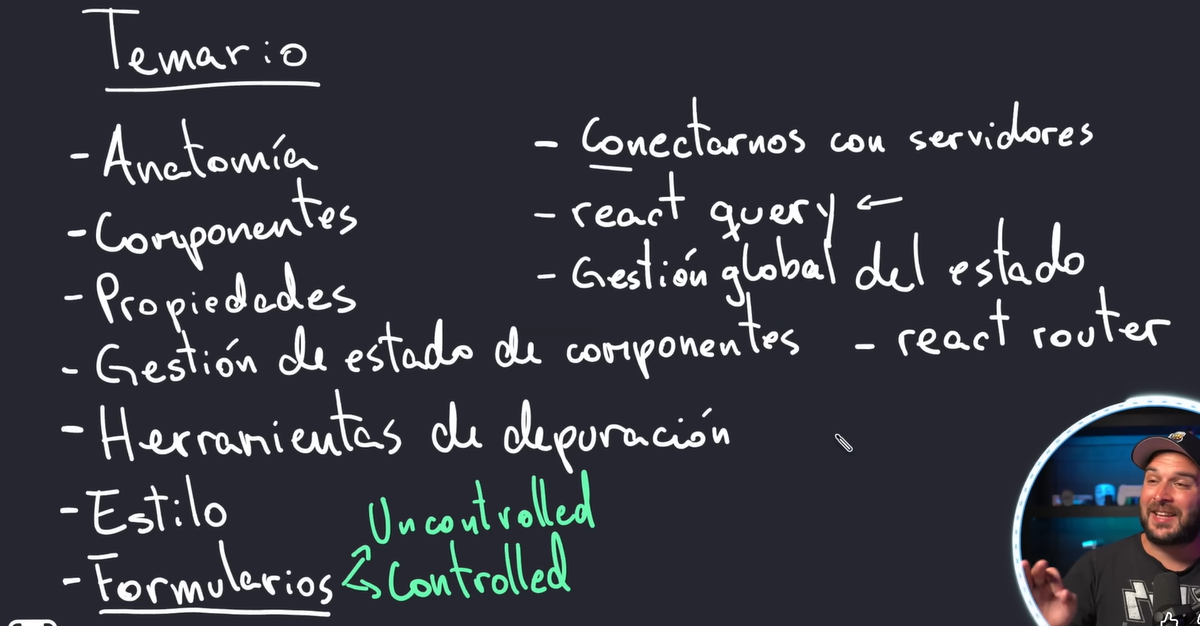
También ocupa el segundo lugar de la tecnología web más usada.



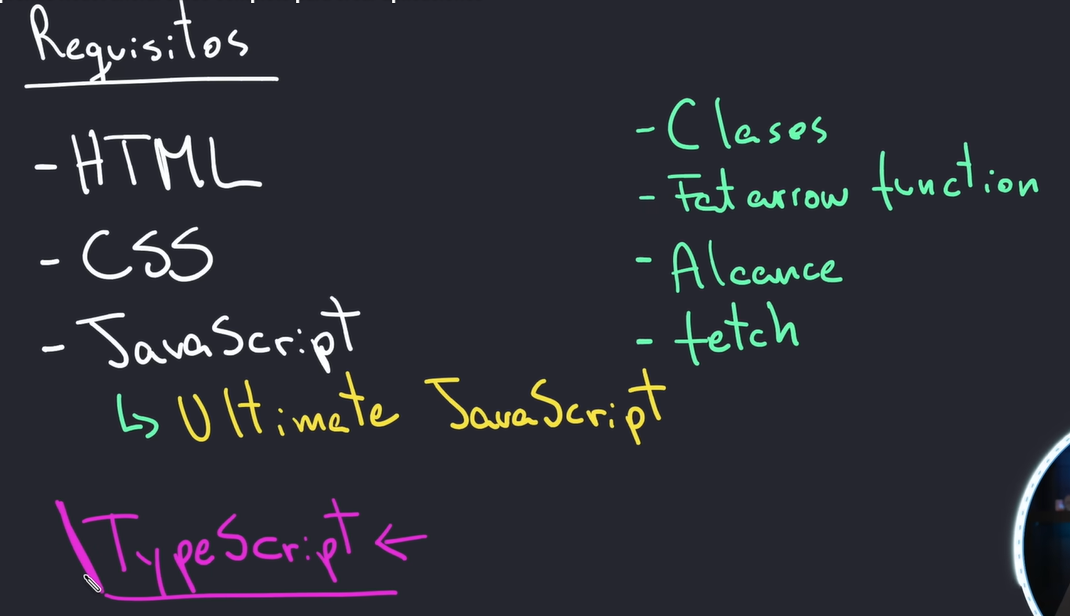
Con react podemos construir aplicaciones web, aplicaciones móviles y aplicaciones de escritorio, para Windows, Linux y Macos.

Además, toda la programación de nuestra aplicación, como por ejemplo conectarnos con servidores (Apis), realizar cálculos o almacenas datos podemos hacerlo con JavaScript.

### **Temario**



### **Requisitos**



TypeScript No es necesario para react.

### **Qué es React ?**

<https://es.react.dev/>

Podemos ver que nos dice:



React es una biblioteca podemos construir aplicaciones web y también aplicaciones móviles para Android y Ios.

También existen otras bibliotecas o frameworks que nos van a permitir junto con react construir aplicaciones para Windows, Macos y Linux a esto es lo que refiere con **native**.

Sin embargo, el framework más popular para construir aplicaciones nativas es react native con este se puede construir aplicaciones para Android y IOS pero el enfoque solamente es aplicaciones móviles ósea para teléfonos Android y Iphone.

React es la biblioteca más popular actualmente en el mercado, por lo que si queremos trabajar como fullstack o frontend aprender react es si o si.

#### Por qué react ?

Cada vez que nosotros creamos una aplicación, en este caso una aplicación web tenemos que trabajar con el DOM (Document Object Model).

**DOM (Document Object Model):**

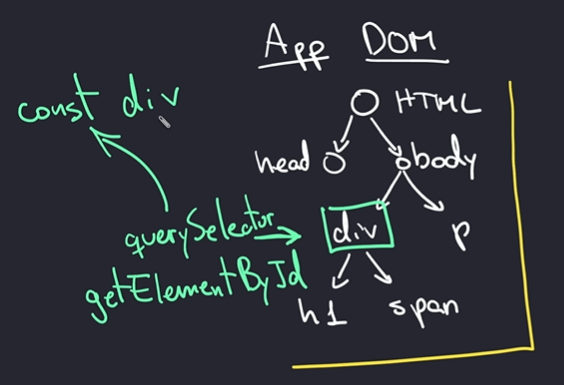
El DOM es un árbol el cual contiene etiquetas, donde partimos con la etiqueta HTML y este va contener las etiquetas de Head y también la de body, dentro la etiqueta de body nosotros podemos tener div también podemos tener párrafos, dentro de estos div podemos tener por ejemplo un h1 y una etiqueta de spam y así sucesivamente.



Este árbol que estamos viendo representa el código HTML de nuestra aplicación.

Con JavaScript nosotros lo que hacemos es ir a buscar alguna de estas etiquetas con por ejemplo el método de querySelector o también podría ser getElementById, después de estos métodos nosotros podemos mantener referencias de cada uno de estos elementos.

De esta manera si nosotros usamos por ejemplo querySelector para poder obtener un dicho div, lo podemos almacenar dentro de una constante lo cual se puede llamar div. Y luego si nosotros queremos actualizar su contenido tenemos que hacer referencia a esta constante.



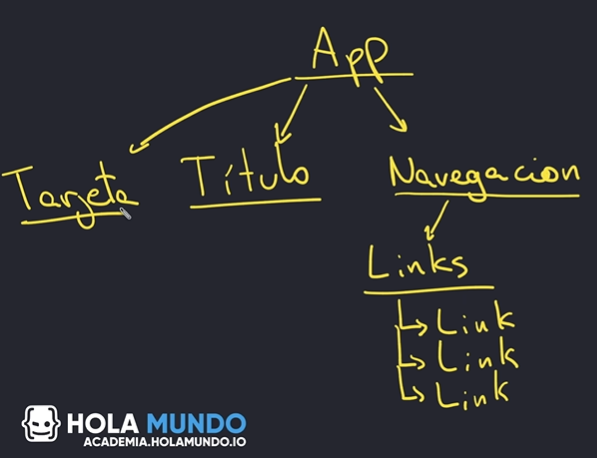
Pero si nuestra aplicación crece y tiene múltiples nodos, mantener actualizado el contenido de nuestra aplicación web utilizando solamente JavaScript se convierte en la tarea repetitiva, tediosa y muy difícil de seguir.

Es por esto razón que se creó la biblioteca de React, react es la biblioteca que se va encargar de actualizar la interface del usuario, nosotros No debemos de ir a buscar cada uno de los nodos para poder actualizarlos, donde esto es tarea de react.

Donde nosotros vamos a tener que encargarnos solamente de describir cómo se va ver nuestra aplicación en determinado momento, pero nunca vamos a tener que encargarnos de ir a actualizarla al DOM.

#### Componentes

Con react nosotros debemos crear pequeños componentes que vamos a poder reutilizar que finalmente nuestra aplicación se va ver más o menos de la siguiente forma:



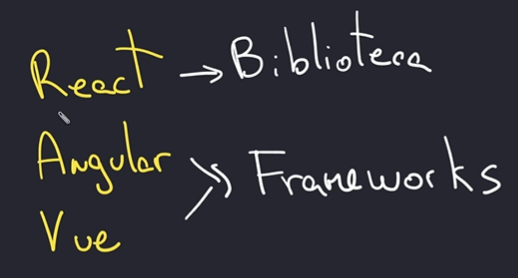
Vamos a tener un componente padre que se va llamar APP, debajo de este podemos tener componentes como Título, Navegación, dentro de navegación podemos tener un componente de Links y dentro de Links podemos tener muchos componentes que va ser referencia a un link en específico, además podemos crear otro componente que se llame tarjeta y así sucesivamente.

De esta manera nosotros podemos reutilizar componentes que hallamos creado y así nuestra aplicación empieza a ser más fácil de mantener porque reutilizamos más nuestro código, **ese es el principal objetivo de react**.

#### Comparación con otros framework

Hay gente que comparan react con otros frameworks. Por ejemplo:

React con angular y con view



**React:**

React es una biblioteca, que se va encargar solamente de la parte de actualizar la vista de nuestras aplicaciones, ósea solamente lo que usuario va terminar viendo.

* React viene siendo un martillo, no tiene ninguna opinión sobre que herramienta utilizar.

Si queremos aprender a gestionar las rutas podemos utilizar la biblioteca de react router, pero si no nos gusta podemos utilizar podemos elegir otra biblioteca o no podemos elegir adsolutamente nada y eso es porque react no es una caja de herramientas uqe no tenemos opinión sobre que usar con él.

**Angular y View:**

Angular y View es un framework que tienen herramientas como por ejemplo gestión de rutas, gestión de estado, se encargan también de actualizar la vista y así sucesivamente

* Angular y View viene siendo una caja de herramientas, y esta caja de herramientas por supuesto que contiene un martillo

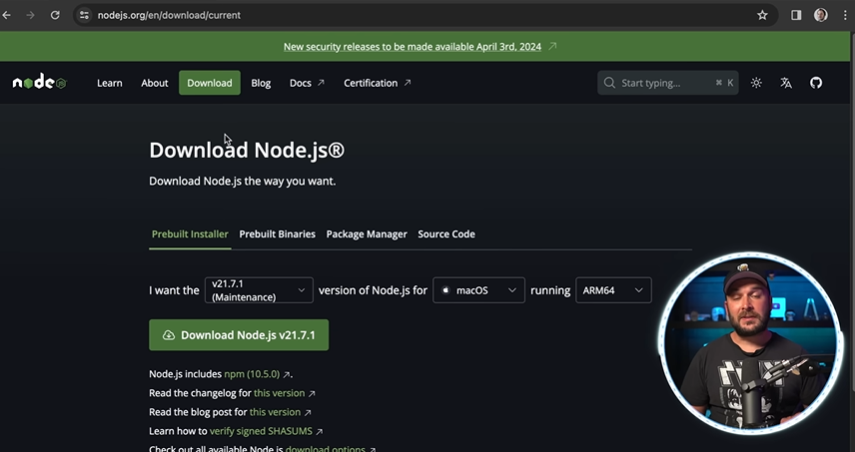


Aquí vamos a ver react y dominarlo a la perfección y luego vamos a ver las bibliotecas más populares que podemos usar con react.

### **Configuración de Ambiente**

#### **Node.js**

Para poder usar react debemos de tener instalado node.js



Vamos a necesitar la versión mayor a la 16, muy importante que sea la versión mayor a la 16



No importa la versión que sea lo importante es que sea mayor a la 16.

#### **VSCode**

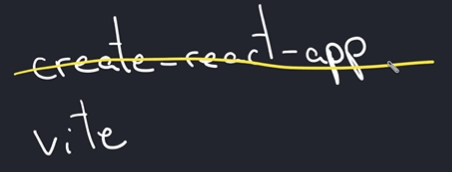
Debemos de tener por lógica el VSCode

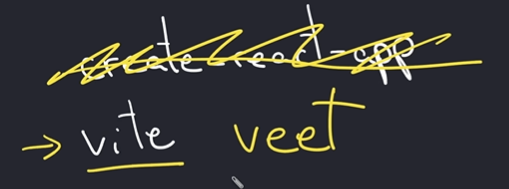
#### **Extensión PrettierX**

Estas extensiones son adicionales, me salteo esta sección del video por el momento no me interesa esta extensión. Prettier formatea el código cada vez que guardemos. Min 11:00

### **Creando una App en React**

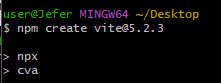
Antiguamente existía una herramienta que se llamaba **create-react-app** esta era herramienta oficial para nosotros aprender a crear proyectos en react, sin embargo, debido a que apareció una nueva herramienta que se llama **vite** lo cual es más rápida y también crea empaquetados más pequeños comparados con **create-react-app** lo cual esta herramienta ya no se usa si no que debemos usar **vite.**





#### **Creación de Nuestro Proyecto**

Damos el siguiente comando para manejar la misma experiencia del master:



A veces pregunta si deseamos proceder, damos **y** que significa si.

Tenemos un error donde no nos deja seleccionar con las flechas es porque **Git Bash no maneja bien programas interactivos** (como los menús donde usas las flechas).

Antes de ejecutar el npm créate vite… Necesitamos usar el comando **winpty**.

**winpty** traduce la entrada del teclado y la salida del programa para que funcione correctamente en Windows.

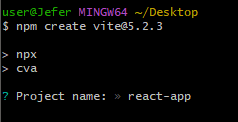


Otra solución es usar consola CMD, ya que Windows cmd funciona perfectamente flechas , menus , etc …

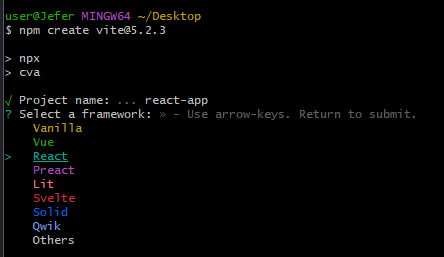
Luego le vamos a indicar un nombre a nuestro proyecto:



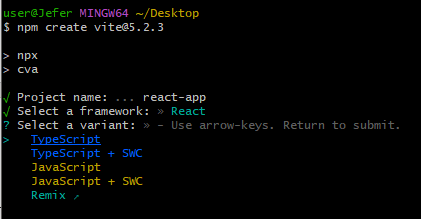
Vamos a ponerle **react-app**, podemos darle el nombre que queramos.



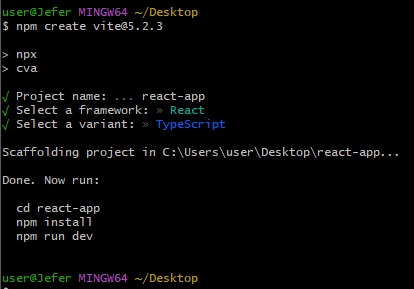
Nos aparece que tipo de proyecto nosotros queremos crear, podemos ver que hay para varios tipos de frameworks en este caso elegimos react:



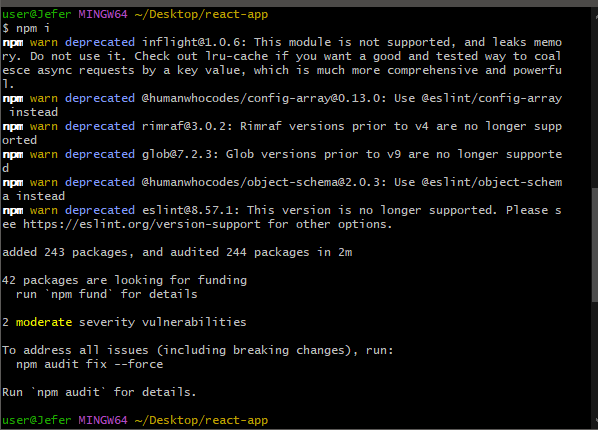
Luego debemos seleccionar la variante o lenguaje en este caso del curso elegimos TypeScript.



Podemos ver que ya creo la estructura de nuestro, ahora necesitamos ejecutar nuestra aplicación donde seguimos todos los pasos que nos indica.



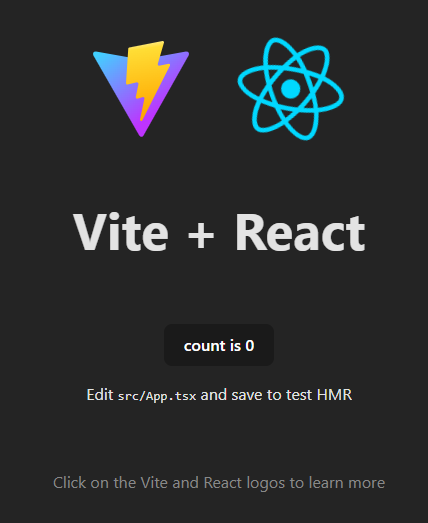


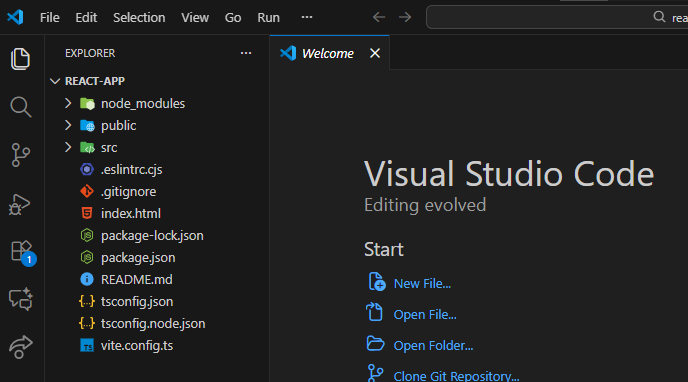


Damos npm i por que la versión corta de npm install es npm i.



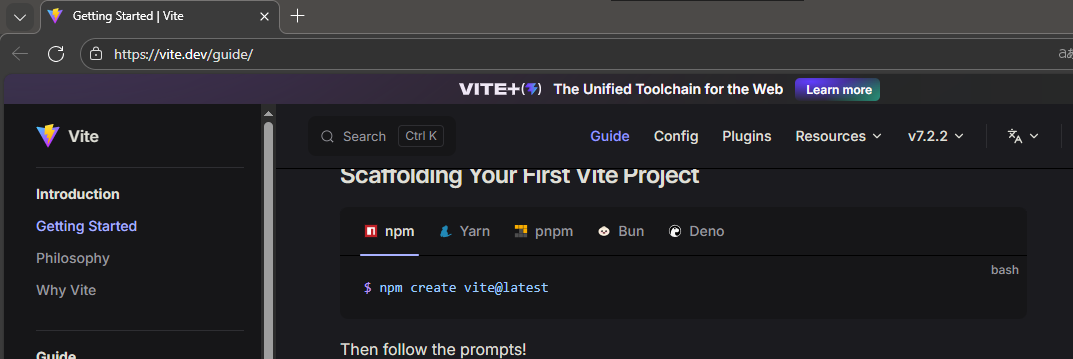
Y listo ya tenemos nuestro proyecto iniciado:



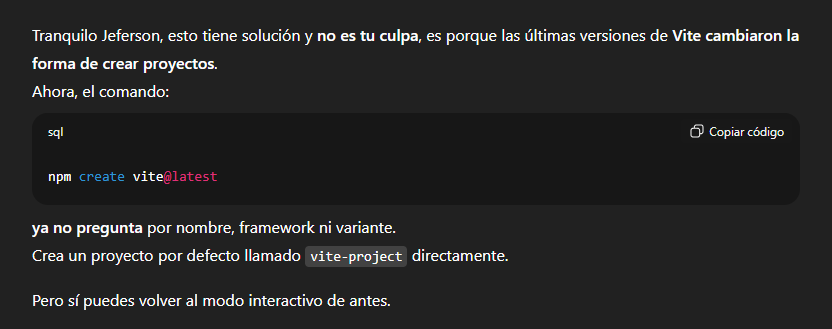


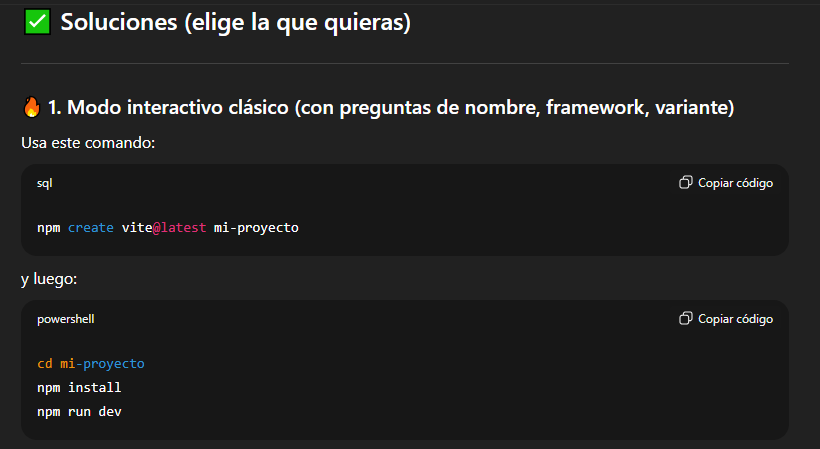
#### **Nota Importante de versión npm create vite@latest y npm create vite@(version)**

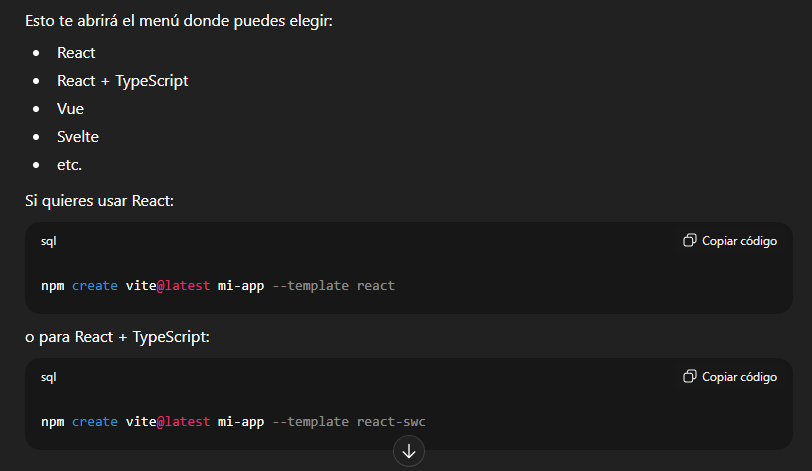
Muy importante que con el comando de la documentación **npm create vite@latest,**

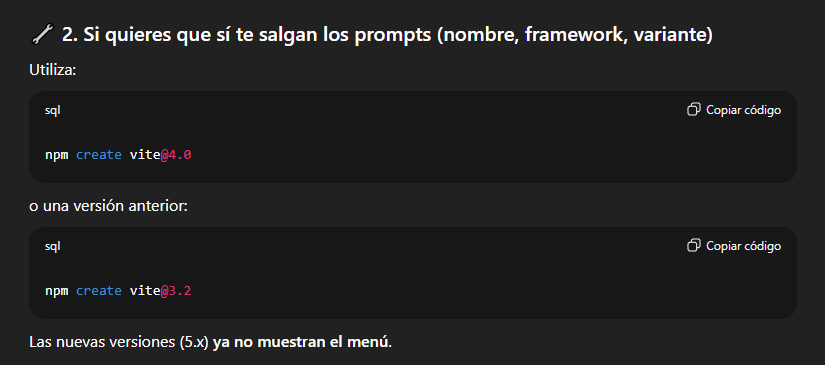


De esta forma vite está realizando contantemente actualizaciones donde el modo de arrancar o de iniciar el proyecto cambia con el tiempo, con **npm create vite@latest** aplicamos la ultima versión y como es la última versión cambia el modo de iniciar o arrancar el proyecto react, preguntar a **chatgpt ejemplo:**

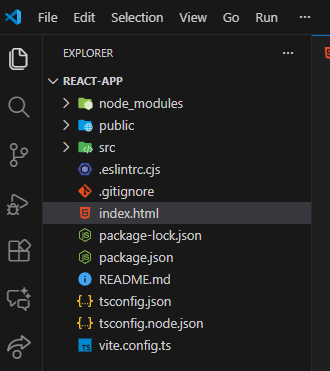




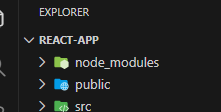




#### **Estructura del Proyecto**

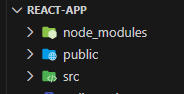


##### Node\_modules:

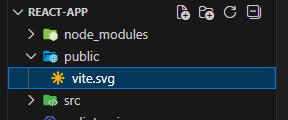


Es el que contiene todas las dependencias de nuestro proyecto aquí se encuentra React, TypeScript y React Dom. **Nosotros nunca deberíamos tocar este directorio.**

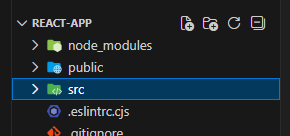
##### Public:



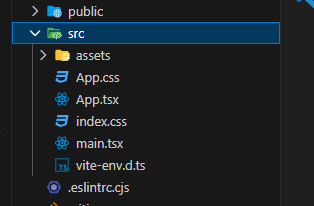
Este archivo se encarga de mantener todos los archivos públicos y que el usuario puede acceder por ejemplo pueden ser las imágenes del logo del proyecto, pueden ser videos o documentos que queremos que el usuario descargue. Podemos ver que solo tiene el logo de vite.



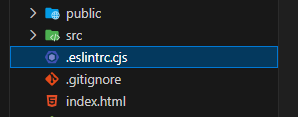
##### src:



Este contiene todos todos los archivos de nuestro código fuente , aquí es donde vamos a pasar todo el tiempo desarrollando.

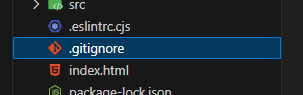


##### .eslintrc.cjs:



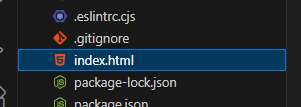
Este archivo no se usa, se usa solo para desarrllo.

##### .gitignore:

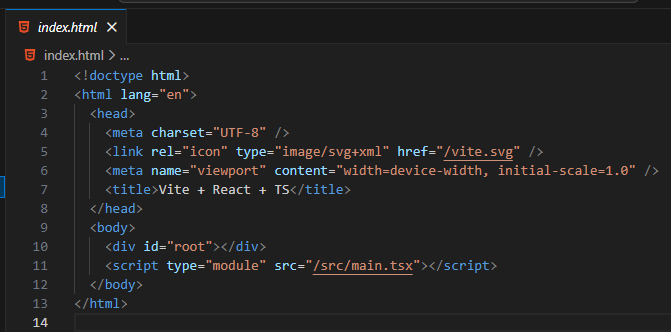


Este archivo es para trabajar con git

##### .index.html:



Este archivo contiene nuestro index principal de nuestro proyecto, lo que realmente nos interesa de nuestro proyecto es el body que allí dentro se encuentra un div.



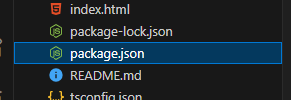
En el div tenemos un id root y tenemos un script donde tiene un main.tsx, este script es el encargado de tomar nuestro código fuente de nuestra aplicación y colocarla finalmente nuestra aplicación en el div id root



Este archivo normalmente no deberíamos tocarlo dentro de body.

Pero si podemos modificar todo lo que este dentro de la etiqueta de head.

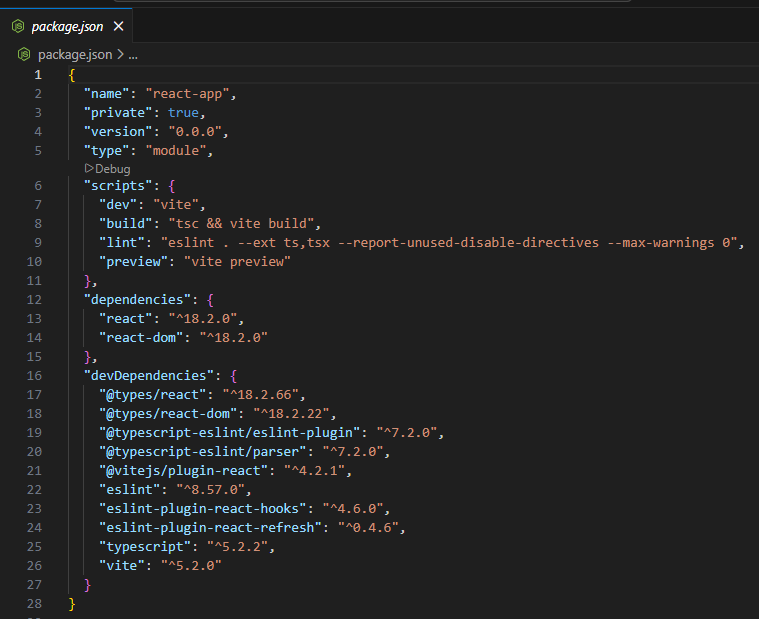
##### Package-lock.json y Package.json:



Este contiene todas las dependencias del proyecto, y también algunos scripts que nosotros podemos ejecutar.

Podemos ver el script dev vite que nosotros ejecutamos cuando damos npm run dev, tenemos build – lint – preview nos sirve para integrar nuestro código hacia un despliegue.

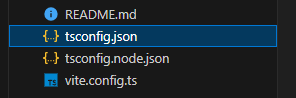
Luego podemos ver las dependencias del nuestra app y las dependencias de desarrollo



##### Readme.md:

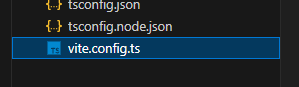
Lo que ya sabemos podemos usarlo para guardar información, meter instrucciones de desarrollo para otros desarrolladores.

##### tsconfig.json y tsconfig.node.json:



Se usa para configurar typescript y cómo se va comportar mientras desarrollemos con vscode, estos archivos no se tocan para nada

##### Vite.config.js:



Este archivo tampoco se toca

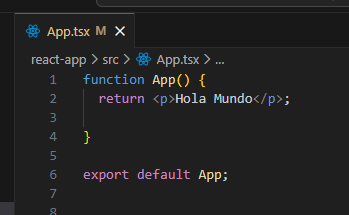
#### **Componentes**

Un componente es simplemente una **función que devuelve JSX**.  
Representa una parte de la interfaz (un botón, un formulario, un encabezado).

Los componentes ayudan a **reutilizar código** y mantener la app organizada.

Un componente es una función que devuelve JSX y representa una parte de la UI.

“Un componente es una función que devuelve JSX y representa una parte de la interfaz. React construye toda la aplicación combinando componentes.”



- Todas las aplicaciones en react necesitan una función que va llamar App, este nombre podría ser otro, así que necesitamos exportar esta función.

- Retornamos el contenido que react le va mostrar a los usuarios y aquí, este contenido podemos ver que hay código HTML dentro de Codigo javaScript, esto es lo que llamamos código JSX, el código JSX es muy parecido a código JSX.

JSX:

JSX = escribir HTML dentro de JavaScript.

Pero NO es HTML, es azúcar sintáctica sobre React.createElement.

Entrevista típica: “¿Qué es JSX?”

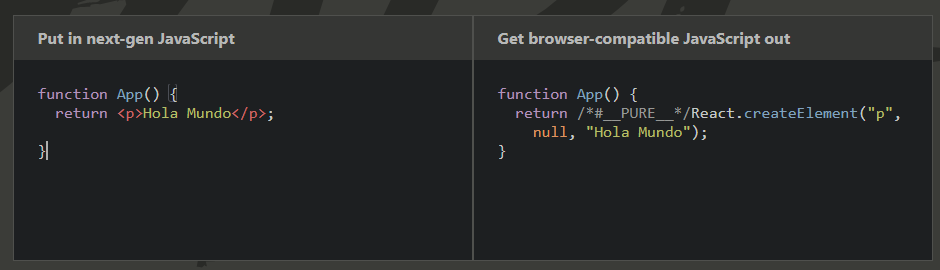
“JSX es una extensión de JavaScript que me permite escribir interfaces como si fuera HTML. Finalmente React lo transforma a funciones JavaScript.”

Este código JSX es transformado a otro código que JavaScript si va poder leer, esto se transforma a **JSX -> React.createElement** y este va tener unas cuantas opciones:

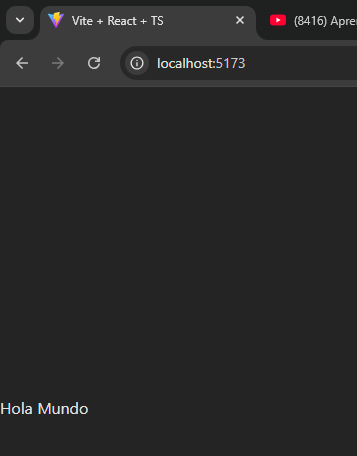
##### Babel.js:

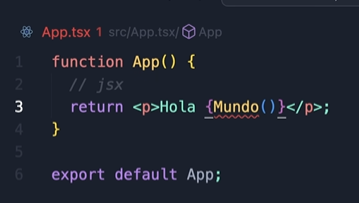
Babel es para ver la como se transforma el JSX a React.createElement.





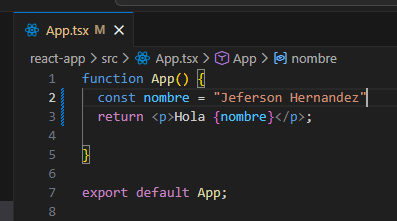
React lo que va hacer es tomar todas estas etiquetas JSX y transformarlas en código JavaScript que va ser legible por el explorador web, esta la razón por la que podemos ver esta etiqueta párrafo que contiene el texto de hola mundo dentro del explorador web.

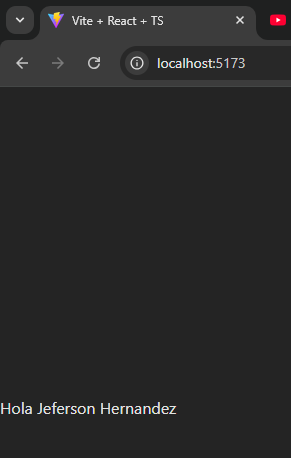




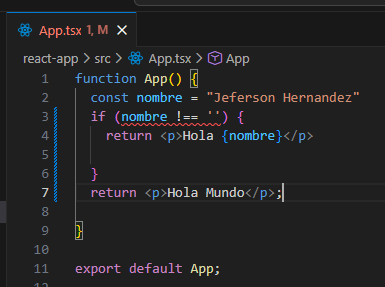
Dentro de esta etiqueta podemos poner una variable , ejecutar una funcion , etc .

En este caso por ejemplo

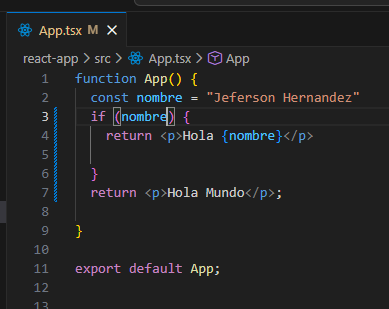




Ahora lo que vamos a poner una validación if

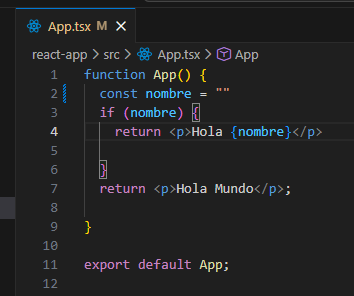


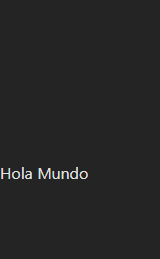
Sale este error para quitar el !== ya que es igual a no ponerlo





Podemos ver que como nos aparece al nombre, pero si le damos un string vacío nos sale Hola Mundo.

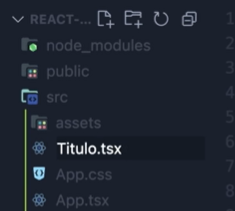


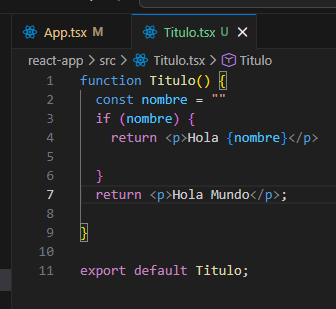


##### Uso de mi primer componente:

Vamos a llevar esta implementación a un componente nuevo para poderlo reutilizarlo:

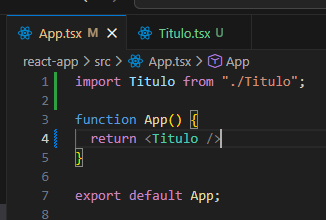
**Creamos un archivo que sea .tsx ya que creamos la app react con typescript, y cuando creamos la app con javascript es .jsx**

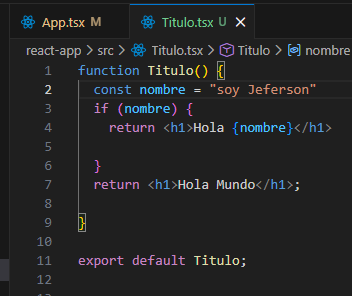




Aquí tenemos nuestro componente creado que se llama titulo vamos a utilizarlo desde nuestro archivo App.tsx.

Luego tenemos que importar aquí en App.tsx nuestro componente de título:







Podemos ver cómo funciona los componentes.

#### **Como funciona React**

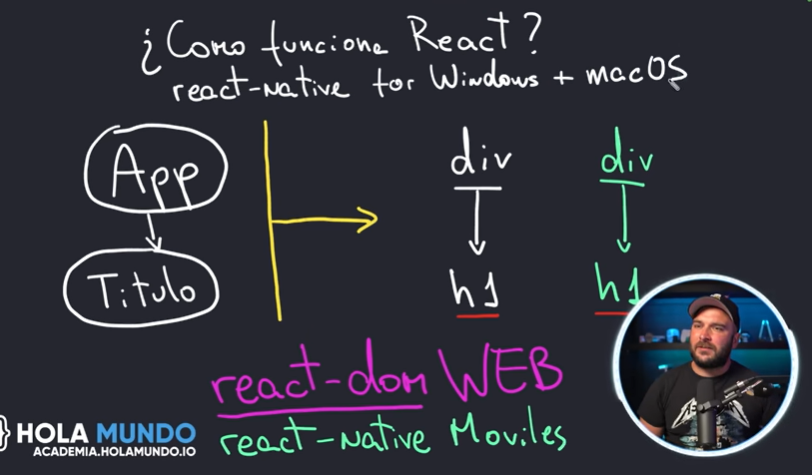
Inicialmente tenemos un componente padre llamado App luego que y debajo de este hay otro componente llamado Titulo, lo que hace react es transformar estos componentes a código HTML generando los árboles o nodos HTML.

Luego cuando existan cambios dentro de nuestros componentes sea en App o Titulo, react va hacer una copia dentro de la memoria del computador(verde), react va buscar cual es la diferencia de todo este árbol que nosotros creamos y luego después de saber dónde se encuentra la diferencia react le va a entregar la responsabilidad a alguien más de poder renderizar esto al explorador web del usuario, **ósea que react no es el responsable de pintar todo el contenido que nosotros escribamos dentro de react, el encargado de pintar todas nuestras aplicaciones en el explorador web se llama react-dom.**

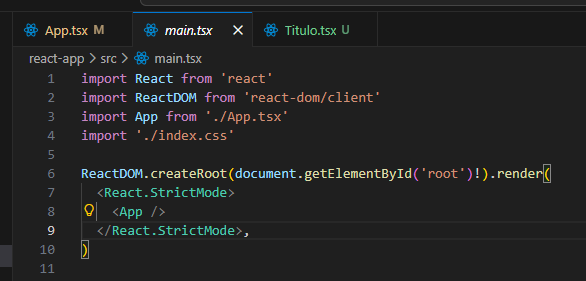
**React-dom**: Renderiza todo el contenido de nuestras aplicaciones. Se usa para construir aplicaciones en la web.

**React-native:** Renderiza todo el contenido de nuestras aplicaciones Moviles. Se usa para construir aplicaciones móviles.

**React-native for Windows + MacOS**: Renderiza todo el contenido de nuestras aplicaciones de escritorio. Se usa para construir aplicaciones de escritorio. Está en desarrollo actualmente y puede contener errores.

****

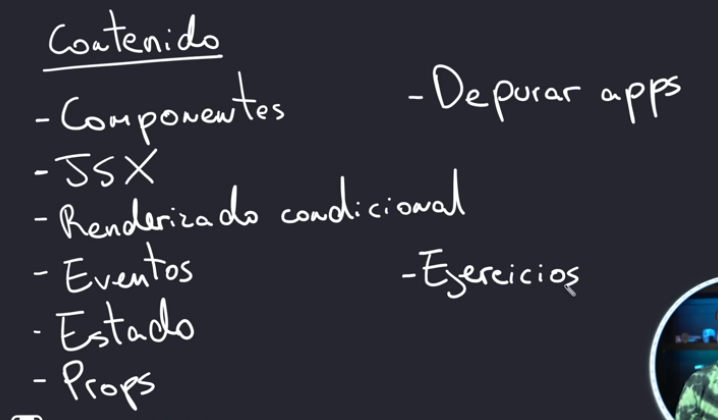
Aquí lo podemos ver el código como funciona react y como lo renderiza en main.tsx:



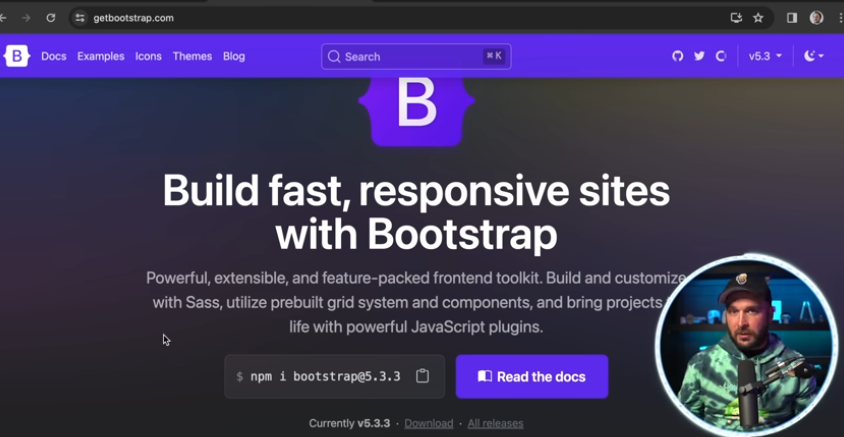
Aquí podemos ver cómo estamos usando react-dom y lo usamos para crear un nodo de react dentro del elemento root, la biblioteca que le estamos pasando es react-dom, luego este se va encargar de renderizar al componente que se llama StrictMode este renderiza nuestros componentes dentro de este tenemos nuestro componente App que es App.tsx.

#### **Introducción**

##### Conceptos Fundamentales de React:



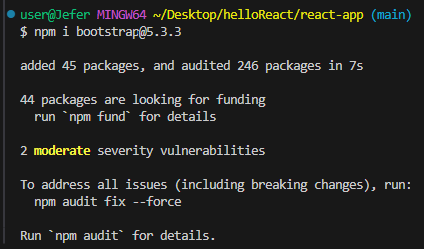
#### **Bootstrap**



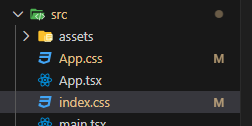
Recordemos que Bootraps nos permita construir herramientas aplicaciones más rápidamente ya que nos entrega estilos y herramientas para construir más rápidamente la aplicación, además puede ser modificado a futuro para que este se adapte más rápido al diseño de la aplicación.

##### Instalación de Bootstrap

Damos el siguiente comando:



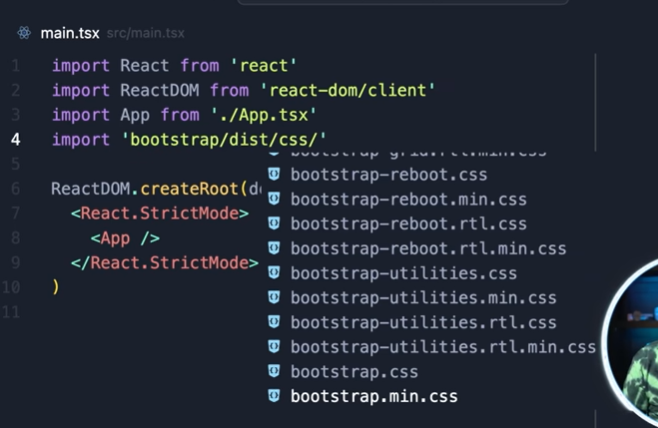
Tenemos dos archivos que nos brinda vite que es App.css y index.css, estos nosotros no lo vamos a usar porque vamos a utilizar bootstrap. Eliminamos todo lo que hay dentro de estos archivos.

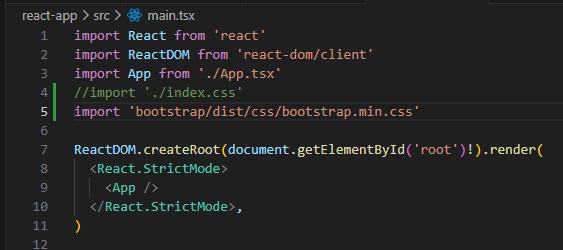


Se actualiza con vite:



Luego realizamos importación de Bootstrap:

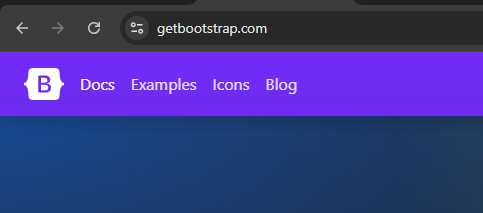




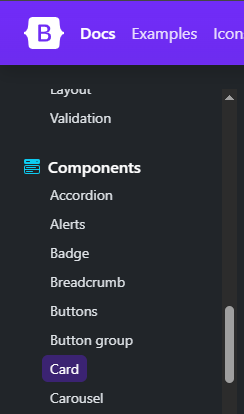
#### **Componente Card**

Vamos a crear nuestro primer componente basándonos en la documentación de bootstrap.

Damos en Docs:



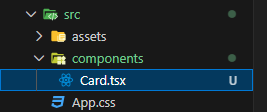
Luego bajamos a la sección de componentes y damos en card:



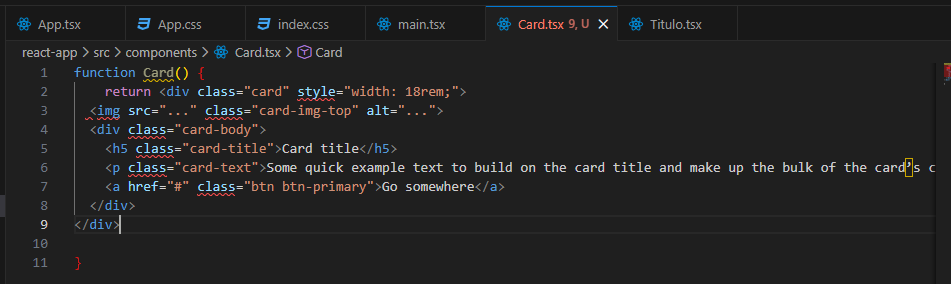
Luego bajamos a las card código que primero encontremos:



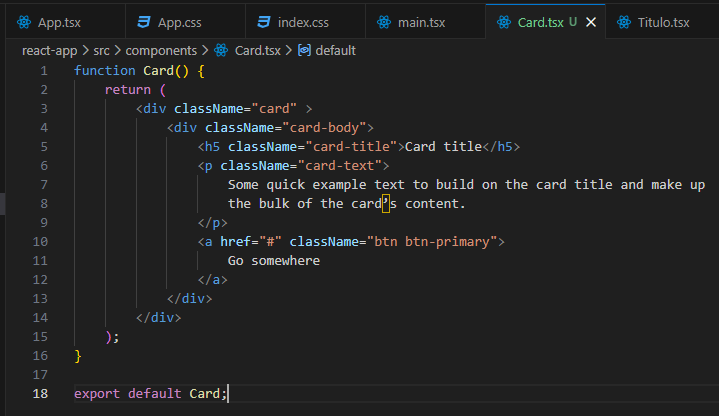
Creamos un nuevo archivo components:



Y creamos nuestro componente y pegamos nuestro código que copiamos de bootstrap:



Y lo adaptamos a JSX:



Luego vamos a nuestro archivo App.tsx para importar nuestro componente de card:



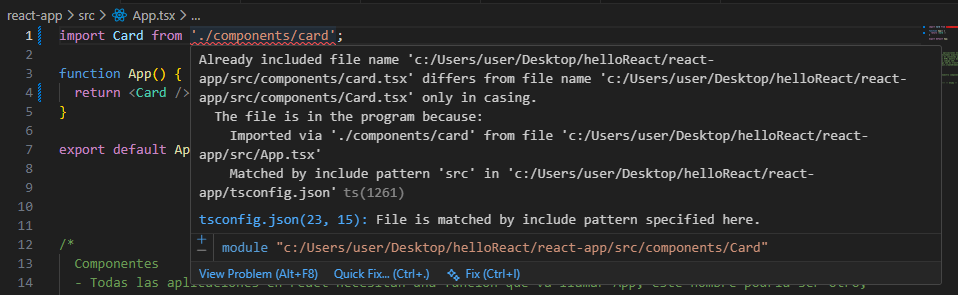
En react no es necesario poner el nombre de la extensión

##### Error de la ruta de importación:

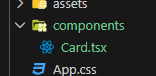
Tenía un error en la ruta de importación en:



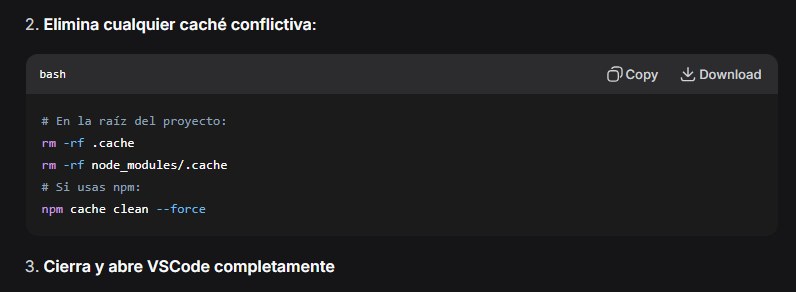
Algo así pero con Card en pascal:



Este error es porque el sistema de archivos de Windows no distingue entre mayúsculas y minúsculas (case-insensitive). Por lo tanto, para el sistema operativo, card.tsx y Card.tsx son el mismo archivo al nombre del archivo.



1. Sistema de archivos de Windows es case-insensitive: Para Windows, Card.tsx y card.tsx son el mismo archivo. No distingue entre mayúsculas y minúsculas.
2. TypeScript/Node.js es case-sensitive: Estas herramientas sí distinguen entre Card y card.



Eliminando el caché de react se solucionó el problema debido a que creo que anteriormente cree el archivo como card.tsx y no como Card.tsx en pascal entonces este caché queda guardado en el node\_modules y hay que eliminarlo. Dejamos el código:

# En la raíz del proyecto:

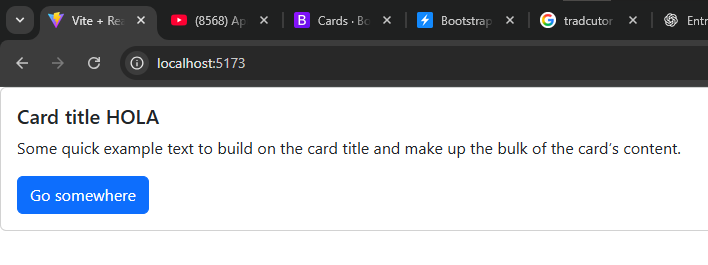
rm -rf .cache

rm -rf node\_modules/.cache

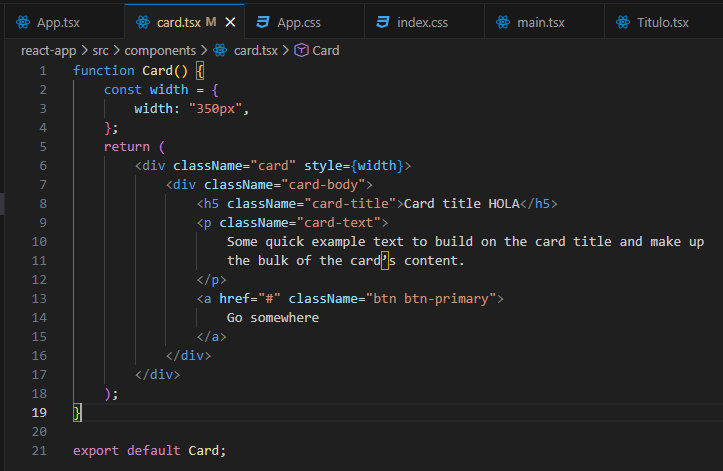
# Si usas npm:

npm cache clean –force

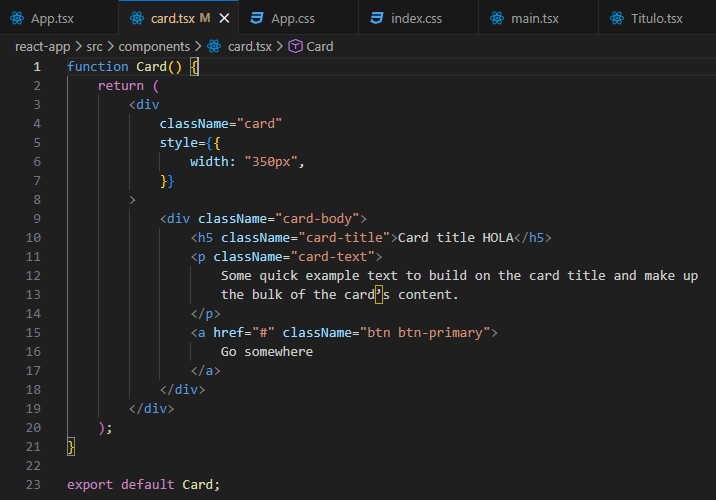
Y finalmente ya podemos ver nuestro primer componente tarjeta:



Vamos arreglar el ancho de esta tarjeta, para hacerlo react no espera un valor string si no que espera un objeto para poder pasarle en este caso un ancho:



Si en el caso el objeto solo tiene una propiedad lo podemos crear dentro de la propiedad de style, de esta manera:



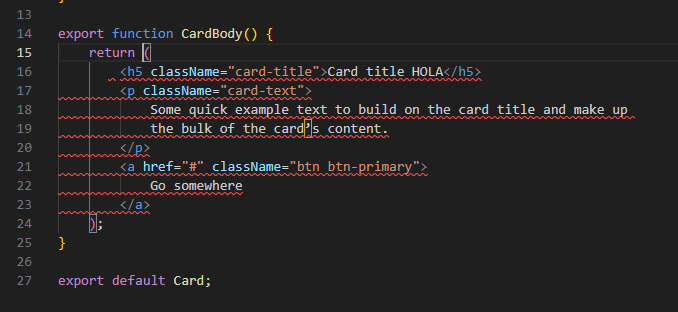
Esto funciona con los dos paréntesis de llave por que el primer paréntesis evalúa una variable dentro de esta y el segundo paréntesis de llaves es para poder definirle que es un objeto. **Esto es una pésima practica mas adelante veremos cómo agregar estilos a los componentes en react.**

Lo más importante es como es como vamos a hacer este componente de tarjeta reutilizable.

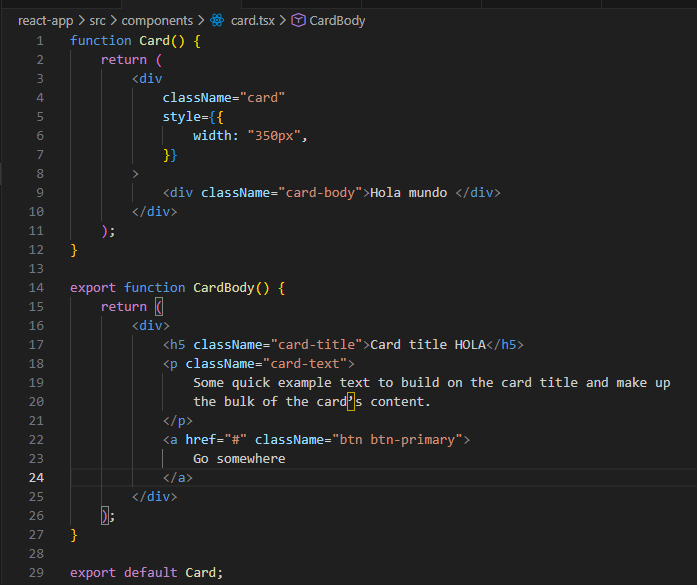
#### **Fragments**

Para hacer más fácil la reutilización de este componente y que todo no sea más difícil, vamos a separar este componente en otros componentes más pequeños.

Por ejemplo si queremos pasarles diferentes estilos y etiquetas al título podemos separar el título, lo mismo con el cuerpo, etc .. Para dejar que quede más separado en otros componentes.



El error consiste en que debemos envolver este componente dentro de otro componente, la solución es envolverlo dentro de un div

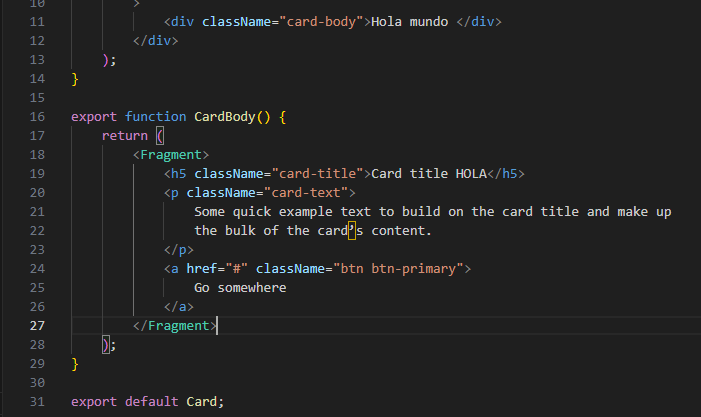


Esto NO es necesario por que se agrega una etiqueta de div que no es necesaria, **este problema lo solucionamos con fragments dentro de react.**

**Para poder usar un fragment lo importamos desde arriba:**

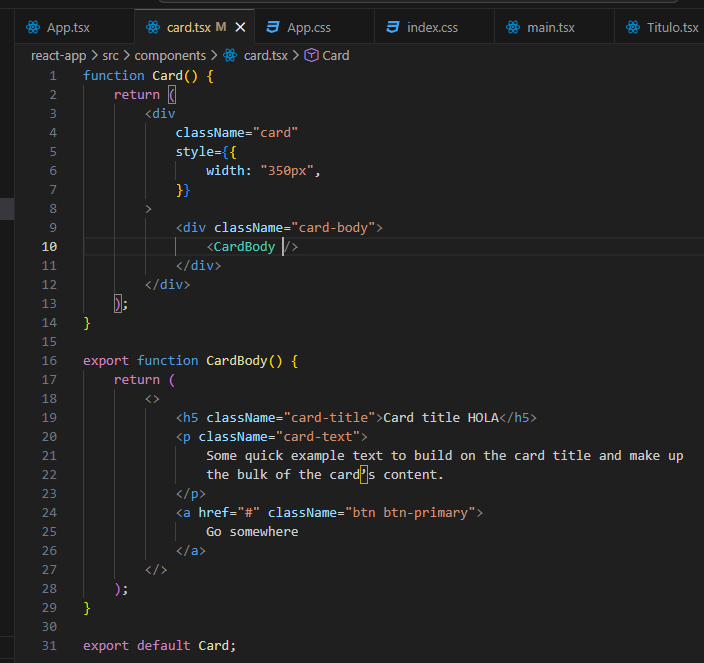
****

Y cambiamos el div por la propiedad Fragment



##### Manera más corta de usar los Fragment:

Eliminamos el nombre de fragment y dejamos las etiquetas que se abren y cierran únicamente sin texto adentro y eliminamos el import.



Esto muy importante porque nos permite separar los componentes dentro del mismo archivo.

Para poder reutilizarlos de otra manera aquí entramos a las propiedades los props.

#### **Pasando Props.**

##### **¿Qué son props?**

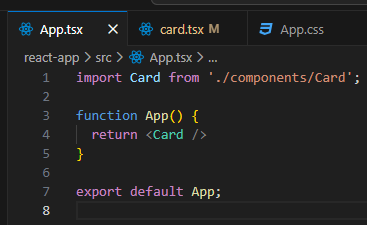
Son los **valores que un componente recibe desde otro componente** (padre → hijo).  
Son **inmutables**, no se pueden modificar dentro del componente hijo.

“Las props son los datos que un componente recibe desde su padre. Son de solo lectura y permiten reutilizar componentes.”

##### **Uso**

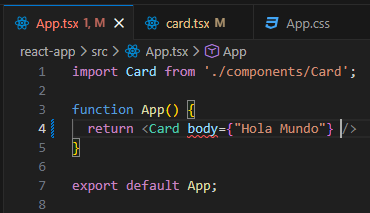
En estos momentos tenemos nuestro componente Card y dentro de este estamos utilizando el componente CardBody.

Vamos al archivo principal que se encuentra utilizando nuestro componente de Card App.tsx:

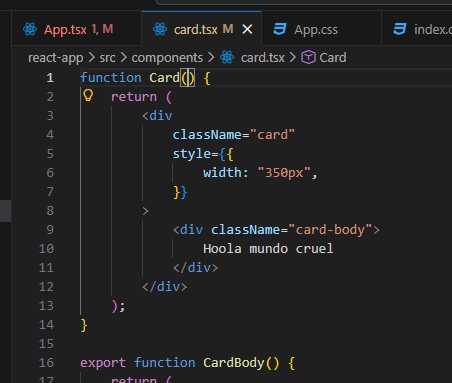


Si queremos pasarle una propiedad para que nuestro componente sea más dinámico lo hacemos aquí mismo donde estamos usando este componente de Card.

En este caso queremos pasarle el body o en este caso en cuerpo de la tarjeta :

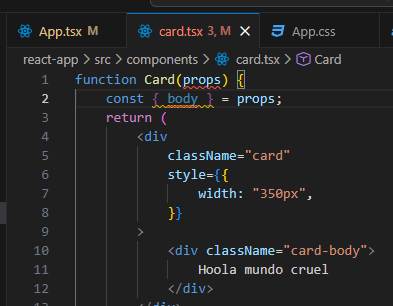


Nos vamos nuestro componente de Card y recordemos bien la propiedad que estamos pasando que llamamos body.

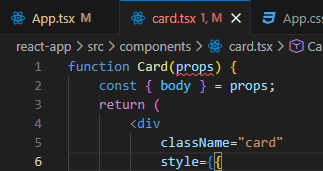


Aquí justo en la definición de la función vamos a recibir todas las propiedades o valores que le pasemos a nuestro componente de Card.

Donde se le entrega el nombre de props y luego cuando queremos obtener todas las propiedades de este componente podemos hacer un object distractorin para poder obtener la propiedad de body .

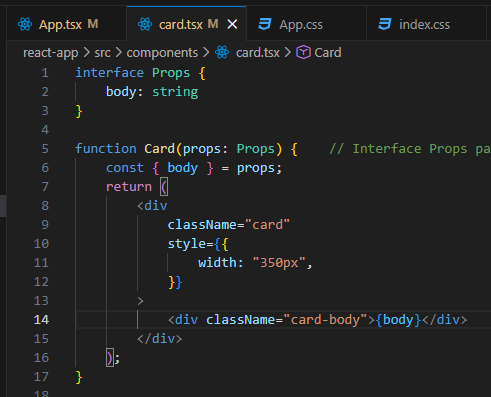


Si este componente tuviese más propiedades lo que podemos hacer es irlas agregando con coma una a una.



Este error debemos indicarle cuales son las propiedades que este va a recibir en este caso va ser un body de tipo **String.**

**Para esto debemos definir una interface,** donde viene siendo la forma que queremos que tenga este parámetro de nuestro componente de Card, podemos ponerle el mismo nombre de props, luego debemos indicarle que lo que va a recibir en este caso va a recibir una propiedad que se llama body y este va tener el valor de String y luego le en el parámetro de rops le pasamos el Props de la interface.

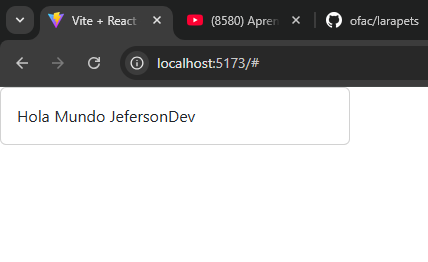


Así le indicamos que este parámetro props va tener este tipo Props donde es un objeto con la propiedad de body.

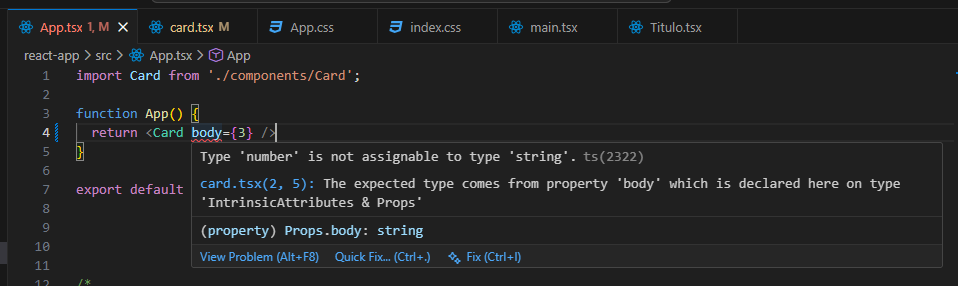
También nos muestra todas las propiedades que va tener este objeto :



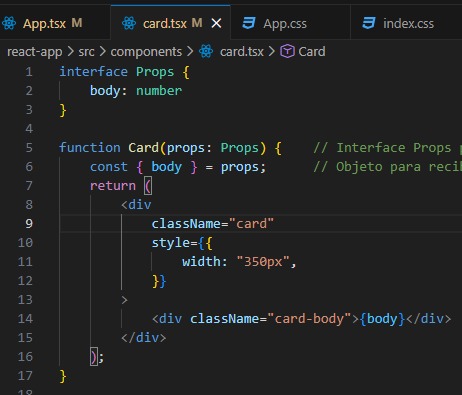
Finalmente podemos ver como pasamos un componente a otro componente, podemos ver cómo estamos pasando propiedades el componente.

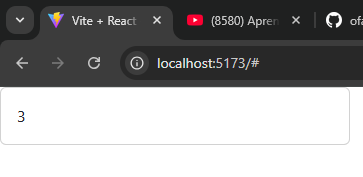


**Nos debemos a nuestro archivo componente App.tsx podemos ver que aquí repasamos a body un string, sin embargo, un string no es necesariamente lo que tenemos que pasar podemos pasar tipos de datos de número, objetos, funciones, incluso otros componentes.**

****

En este caso si le asamos un tipo de dato numérico nos sale error, or que debemos actualizar el tipo de body por que le indicamos que era un string. Por ejemplo, lo modificamos y nos da:

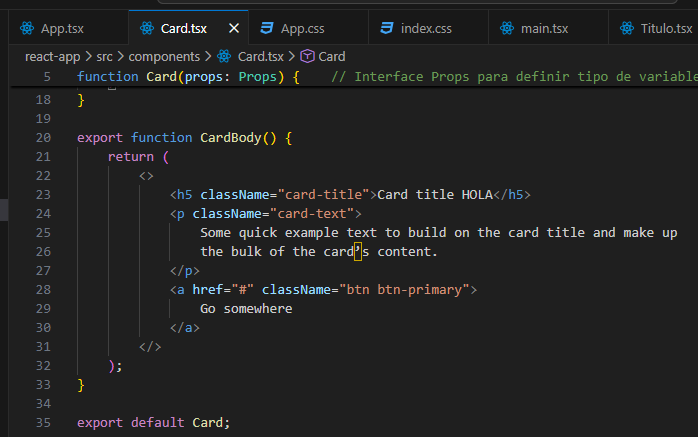




Esto es una grande ventaja de TypeScript, por que nos indica el error del tipo de dato que estamos enviando o pasando.

#### **Múltiples Props.**

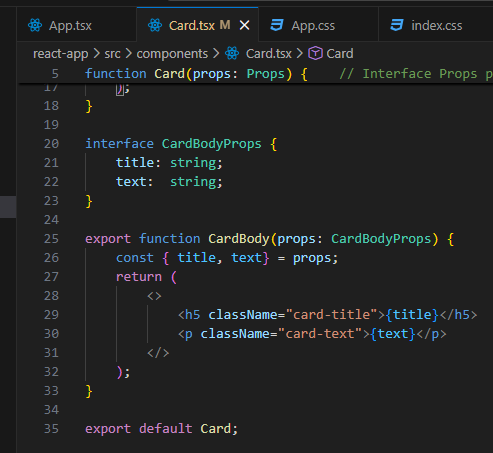
Vamos para el componente CardBody, este componente podemos ver que necesita un título y necesita un texto.



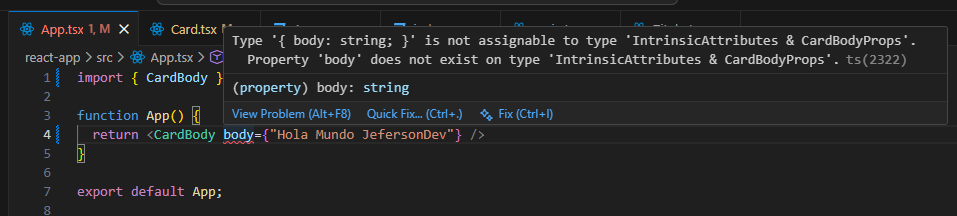
Aquí lo que nos interesa es recibir el título y el texto.

Lo que vamos a hacer es que este componente CardBody reciba propiedades del título y del texto.

Realizamos primero para **recibir** **las propiedades** de título y texto:



Ahora en el componente App.tsx que se **envía las propiedades** vamos a utilizar este componente CardBody en lugar de Card:

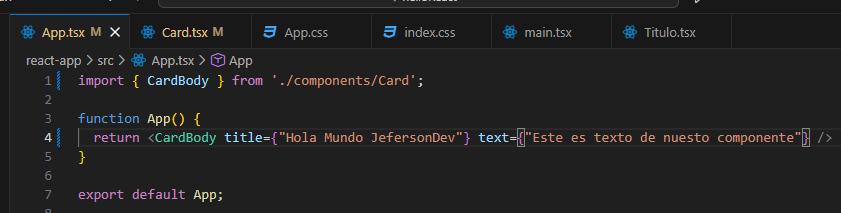


Nos dice que la propiedad body no existe en CardBody, lógicamente tenemos que agregar nuestras propiedades que recibe que es title y text.

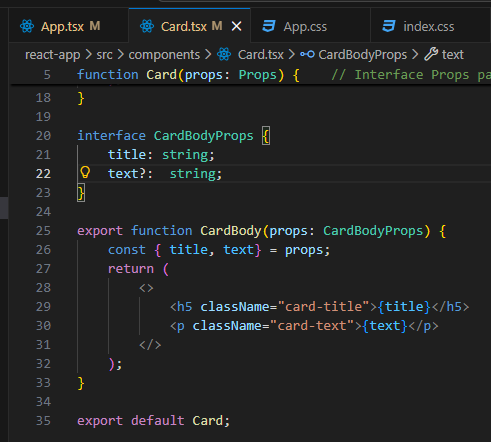
**Muy importante que debemos indicarle todas las propiedades que estamos enviando si no nos aparece error.**



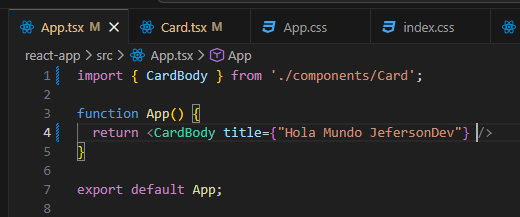
Quedando de la manera:

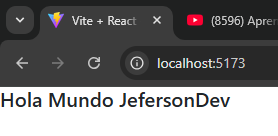


**Pero si nosotros queremos que usar solo una propiedad y que texto por ejemplo sea opcional y no salga error y que estas propiedades no siempre se las tengamos que enviar.** Nos devolvemos a la definición del componente en la definición de su interface y le damos a la que queremos que sea opcional el ? queremos text que sea opcional:

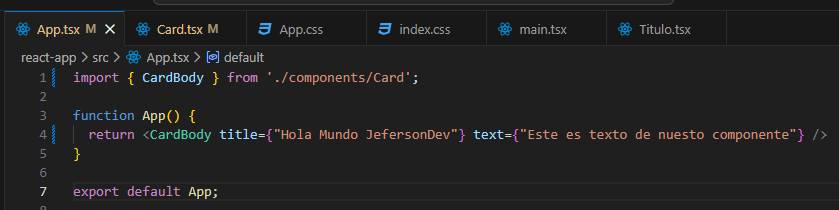


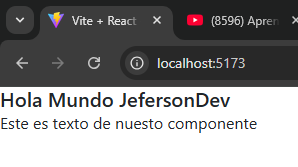
Nos devolvemos al que envía las propiedades App.tsx y borramos la propiedad text porque es opcional que ya la enviemos.





Y si lo ponemos de nuevo el text:





Nos funciona perfectamente.

Pero aquí hay un problema, aquí le estamos pasando propiedades a este componente de CardBody pro finalmente el componente de CardBody no es el que queremos utilizar si no el de componente de Card, hasta ahora vamos a ver una forma en que nosotros vamos tomar un componente que ya existe y pasárselo a otro

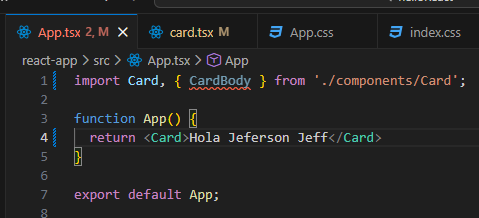
#### **Children**



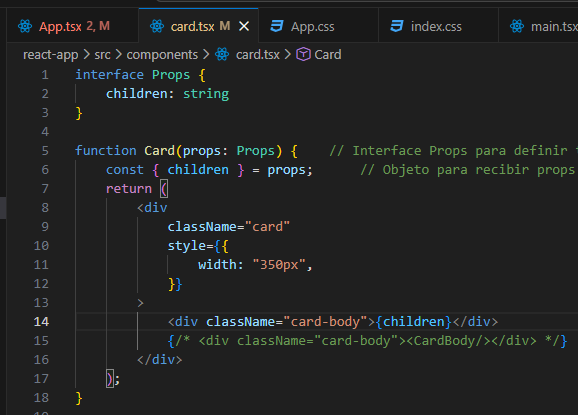


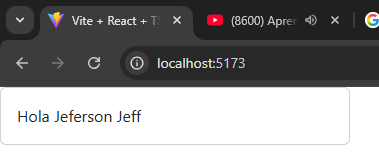
##### **Segunda Forma de Pasar props**

Esta no es la única manera de pasarle propiedades a nuestro componente de card, hay otra forma que es con la propiedad de children.



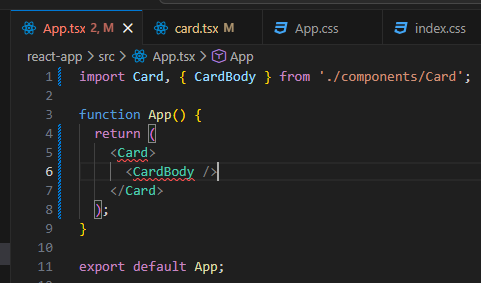
Cambiados la propiedad de body por children:



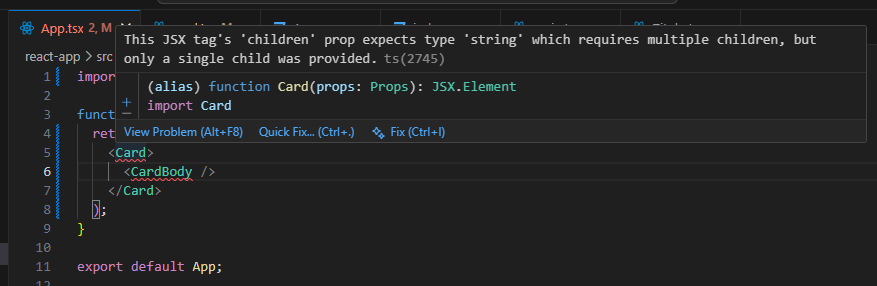


##### **Pasando Props Complejos Como Componentes**

Ahora vamos a pasar una estructura más compleja a este componente por ejemplo si le queremos pasar código HTML, o etiqueta de párrafo, o si le queremos pasar un componente directamente:

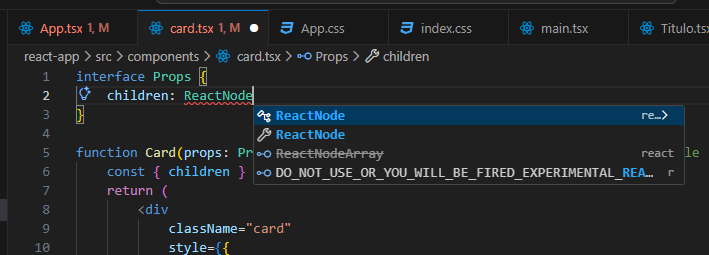


Pero sale un error diciendo que la propiedad childrel espera un tipo string.



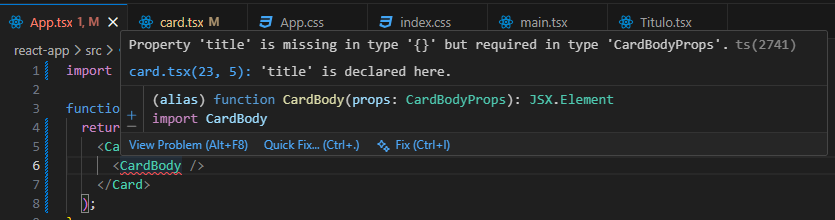
Pero nosotros no le estamos pasando una propiedad de tipo string si no una propiedad componente CardBody.

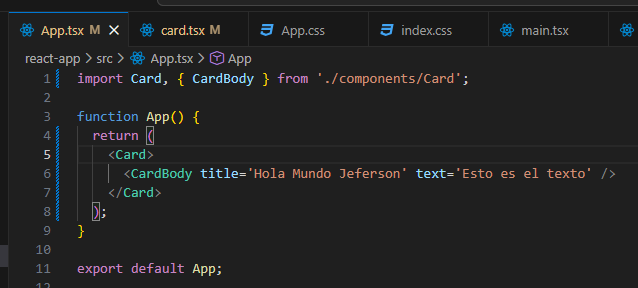
Para esto cambiamos el tipo string por ReactNode, donde nos realiza una importación:

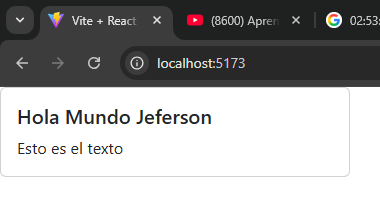




Y luego le mandamos nuestro props o propiedad a nuestro title y text de nuestro CardBody:



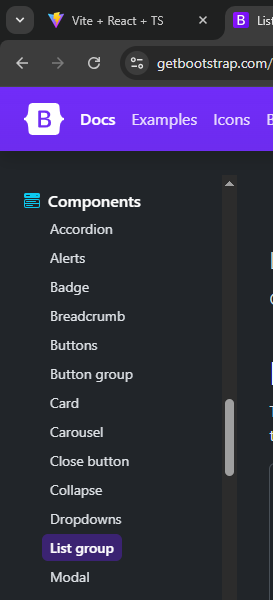


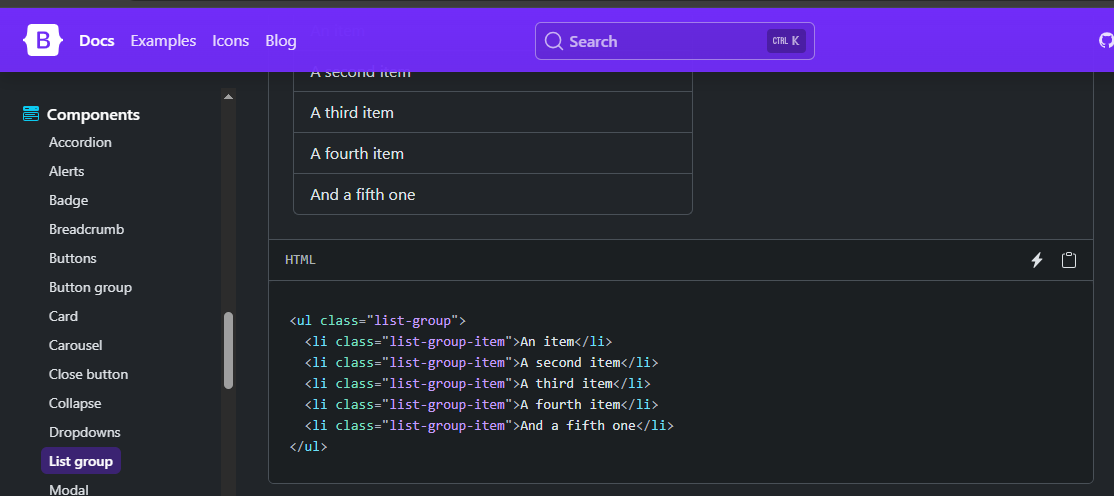


Aquí vimos el cómo poder crear nuestro componente y a ese componente pasarle más componentes, de esta manera podemos ver cómo podemos utilizar los porps o propiedades y también la propiedad especial children que se pasa de esa manera.

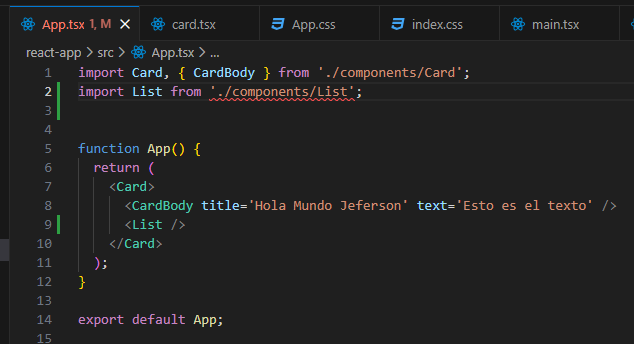
#### **Imprimir Listas**

Vamos a ver cómo podemos imprimir una lista de componentes, vamos a bootstrap:

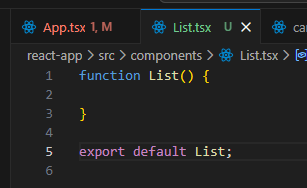




Creamos otro componente List en nuestro App.tsx:

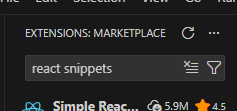
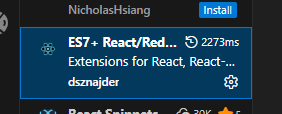


Creamos nuestro componente en List.tsx, cada vez que nosotros creamos un componente debemos escribor esto una y otra vez :



Para esto vamos a instalar una extensión que nos permite auto completar:

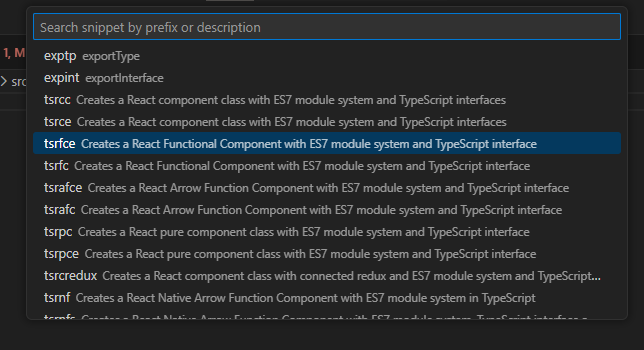
##### **Extensión de Autocompletado**

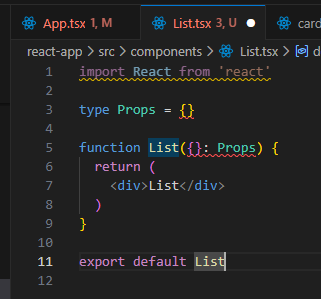


Presionamos para acceder a todos los snippets de react:





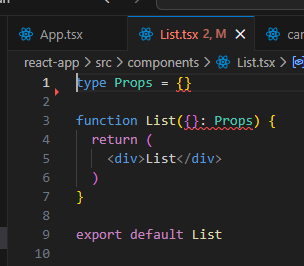
Seleccionamos y nos aparece automáticamente todo el código que necesitamos:

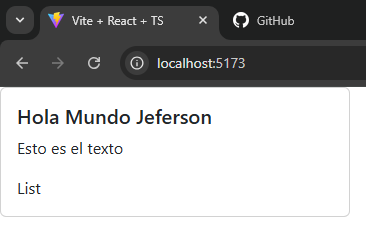


Nos crea el mismo componente según el nombre del archivo.

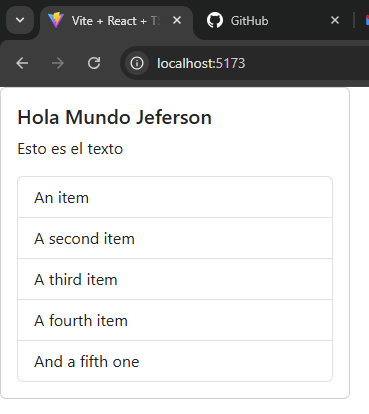
Podemos ver que nos crea el **type Props** donde anteriormente donde también se puede realizar con **interface** como lo hicimos anteriormente mente en el componente de Card

Y ya podemos ver que nos muestra nuestro componente de List:

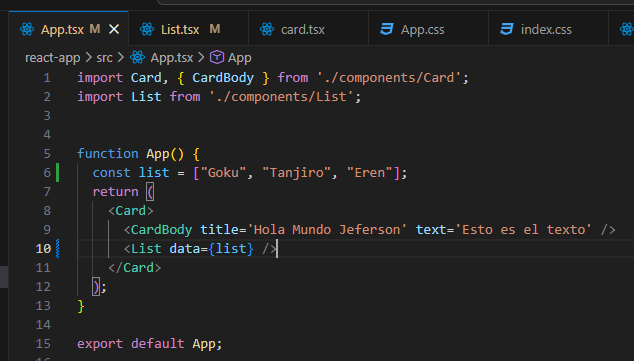




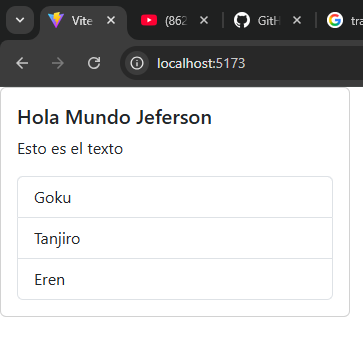
Copiamos nuestro código de bootstrap y lo arreglamos:



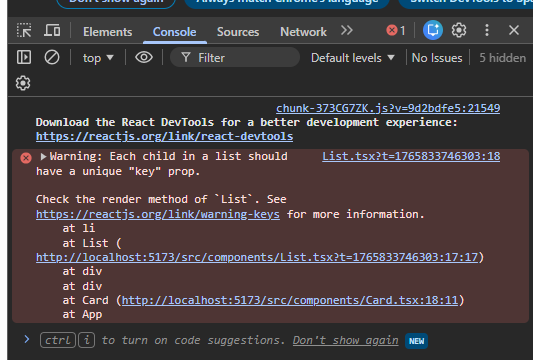
Aquí lo que queremos es No tener una lista fija, si no pasarle un listado a nuestro componente de List:





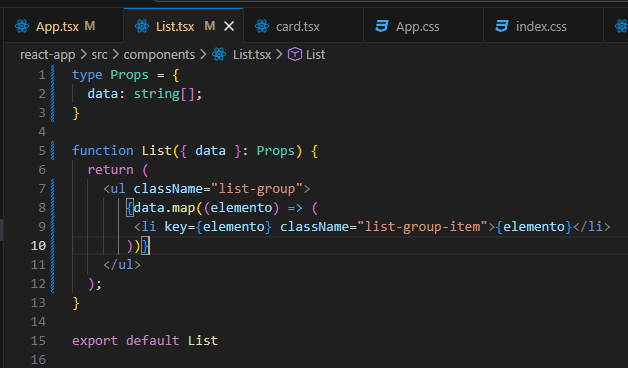


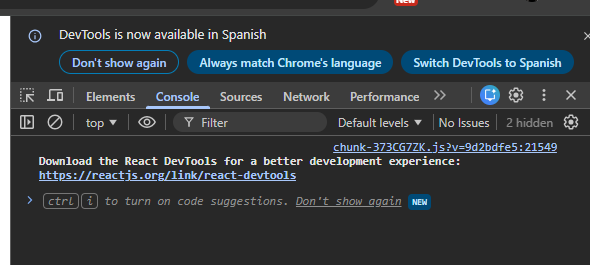
Podemos observar que tenemos un error en el inspeccionador:



Este error nos indica que cada uno de los hijos en un listado tiene que tener un identificador único que se llama key, donde por lo general cuando nosotros nos traemos los datos de una base de datos los elementos van a venir con una propiedad de ID, pero en este caso como los datos no vienen de un BD vamos a utilizar el nombre para asignarles un identificador único:

Le damos la key como el nombre del elemento:





Y se nos quita el error.

Recapitulando esta sección vimos como desde un componente padre App.tsx, nosotros podemos crear datos como por ejemplo una lista y pasarlos a otro componente en este caso List y dentro de este vimos como iterar los elementos que le estamos pasando para poder imprimir un listado.

Minto: 1:03