



REACT

clase n° 10

*REC



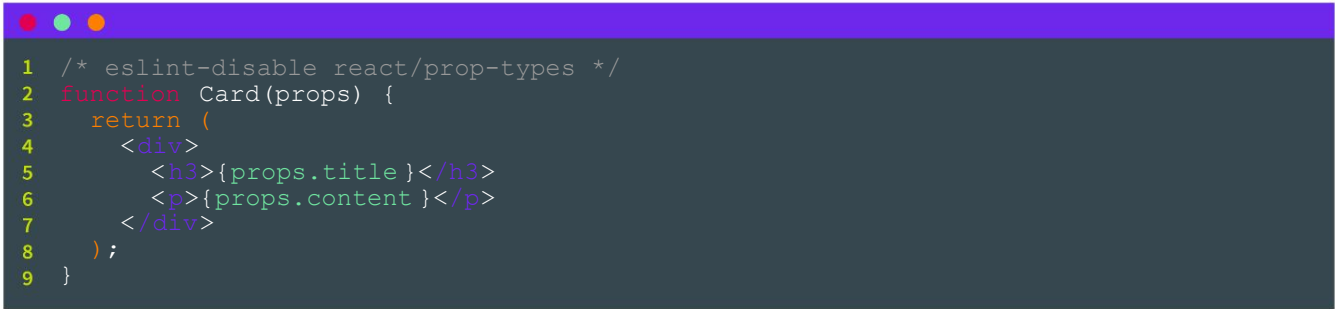
recordá
poner a grabar la clase

importante

Cuando trabajamos con props en react usando vite, nos va a saltar un error de eslint que nos dice que no hemos definido las prop-types. PropTypes es una forma de validar las propiedades (props) que se pasan a los componentes en React. Se utiliza para verificar que las props recibidas tengan el tipo de dato esperado y, opcionalmente, que sean obligatorias. Esto ayuda a identificar errores durante el desarrollo y a documentar las props que espera un componente.

hay 3 formas de solucionar este problema ➡

1. *Deshabilitar la regla de eslint que nos está dando problemas. Para ello, podemos agregar un comentario en la parte superior del archivo que nos está dando problemas. Si es en mas de uno, se deberá agregar en todos.*




```
1  /* eslint-disable react/prop-types */
2  function Card(props) {
3    return (
4      <div>
5        <h3>{props.title}</h3>
6        <p>{props.content}</p>
7      </div>
8    );
9  }
```

- 2.** *Importar PropTypes de la librería prop-types y definir las propiedades de nuestro componente. También, como en el caso anterior, se deberá agregar en todos los archivos que nos estén dando problemas. Por ejemplo:*

```
1 import PropTypes from 'prop-types';
2 export const Componente = ({ name, age, isStudent }) => {
3   return (
4     <div>
5       <h1>{name}</h1>
6       <p>Edad: {age}</p>
7       <p>Estudiante: {isStudent ? 'Sí' : 'No'}</p>
8     </div>
9   );
10 };
11 MyComponent.propTypes = {
12   name: PropTypes.string.isRequired, // Obligatoria y debe ser string
13   age: PropTypes.number,             // Opcional y debe ser número
14   isStudent: PropTypes.bool          // Opcional y debe ser boolean
15 };
```

- 3.** *Agregar la regla "react/prop-types": "off" en el archivo eslint.config.js*
Por ejemplo:



```
1 {  
2   "rules": {  
3     "react/prop-types": "off"  
4   }  
5 }
```

export e import

El sistema de **módulos de JavaScript** (ES6) organiza el código en fragmentos independientes que puedes compartir entre archivos. Esto ayuda a que las aplicaciones sean más modulares y fáciles de mantener. ¿qué quiere decir esto? Que puedes dividir tu código en archivos más pequeños y reutilizarlos en diferentes partes de tu aplicación.

Exportación:

- **export default:** Cada archivo puede tener una única exportación por defecto. Es ideal cuando un archivo se enfoca en una sola funcionalidad, como un componente principal.
- **export named (exportaciones nombradas):** Puedes exportar múltiples funcionalidades en un archivo. Esto es útil cuando el archivo contiene funciones, variables o componentes relacionados.

Importación:

- **default:** Al hacer una importación por defecto, puedes nombrarla como quieras al momento de importarla.
- **named:** Para las exportaciones nombradas, usas exactamente el mismo nombre definido en el archivo original, y debes encerrarlo entre llaves ({}).

```
1 // $ Button.js
2 // Exportación default
3 export default function Button() {
4   return <button>Default Button</button>;
5 }
6 // Exportación named
7 export const SecondaryButton = () => {
8   return <button>Secondary Button</button>;
9 };
```



```
1 // App.js
2 // Importación default
3 import Botoncito from './Button';
4 // Si importamos de esta manera, se renderizará el botón por defecto
5 // La función Button() que se exporta por defecto en Button.js
6 import SecondaryButton from '../components/Button';
7 // Si importamos de esta manera, lo que sucederá es que
8 // se renderizará el botón por defecto, ya que no se está importando el botón
9 // secundario
10 // Importación named
11 import { SecondaryButton } from './Button';
12 // Si importamos de esta manera, se renderizará el botón secundario
13 function App() {
14   return (
15     <div>
16       <Botoncito />
17       <SecondaryButton />
18     </div>
19   );
20 }
21 export default App;
```

react.fragment

En React, los componentes pueden devolver un único elemento o nodo. Por lo que si intentamos hacer un return de 2 elementos, no funcionará. Podríamos pensar en devolver varios elementos dentro de un `<div>` Sin embargo, hay casos donde no necesitas un contenedor real como `<div>`. Usar un contenedor innecesario puede agregar nodos al DOM, lo que puede causar problemas de diseño o reducir el rendimiento.

React.Fragment soluciona esto al permitir agrupar elementos sin añadir un elemento visible en el DOM. Su abreviatura `<>...</>` es funcionalmente idéntica y más concisa.

```
1 export const FragmentExample = () => {
2   return (
3     <>
4       <h1>Fragmento en React</h1>
5       <p>Este contenedor no agrega nodos extra al DOM.< /p>
6     </>
7   );
8 }
```

La forma correcta es la siguiente:

```
1 export const FragmentExample = () => {
2   return (
3     <>
4       <h1>Fragmento en React</h1>
5       <p>Este contenedor no agrega nodos extra al DOM.< /p>
6     </>
7   );
8 };
9
10
```

props y atributos

Las **props** (abreviación de properties) permiten **pasar datos** desde un componente padre a un componente hijo. Estos datos deben ser inmutables: una vez definidos, no pueden ser modificados directamente por el componente hijo. Esto asegura que el flujo de datos sea unidireccional y predecible. Esto quiere decir que si pasamos un array a un componente, no debemos hacer un `array.push()` en el componente hijo, ya que esto modificaría el array original. En su lugar, debemos crear un nuevo array con los datos actualizados y pasarlo de nuevo al componente padre.

Puedes pasar cualquier tipo de dato: strings, números, objetos, arrays, funciones, etc. Las props son accesibles dentro del componente hijo mediante el objeto props.

```
1 // Componente Padre
2 function App() {
3   const tareas = ['Cortar el pasto', 'Ir al supermercado', 'Estudiar REACT!'];
4   // Array de tareas original que se pasa como prop
5   return (
6     <div>
7       <h1>Lista de Tareas</h1>
8       <ListaDeTareas tareas={tareas} /> // Acá pasamos el array
9     </div>
10  );
11 }
12 export default App;
```

```
1 // Componente Hijo
2 import PropTypes from 'prop-types'; // Importar PropTypes
3 export const ListaDeTareas = ({ tareas }) => {
4   // Crear un nuevo array basado en las tareas pasadas
5   // como props, para no modificar el original
6   const TareasFormateado = tareas.map((tarea) => `✅ ${tarea}`);
7   return (
8     <ul>
9       {TareasFormateado.map( (tarea, index) => (
10         <li key={index}>
11           {tarea}
12         </li>
13       ))}
14     </ul>
15   );
16 };
17 ListaDeTareas.propTypes = {
18   tareas: PropTypes.arrayOf (PropTypes.string).isRequired
19 }; // Ejemplo de como usar PropTypes
```

explicación

- El componente padre (App) pasa un array (tareas) como prop al hijo.
- Este array contiene una lista de tareas.
- El componente hijo (ListaDeTareas) recibe el array a través de las props.
- En lugar de modificar directamente el array tareas, el componente hijo usa el método `.map()` para crear un nuevo array (TareasFormateado) donde a cada tarea se le pone un símbolo de verificación (☐)
- El array original permanece inmutable.
- El componente hijo nunca altera tareas. En su lugar, genera un nuevo array basado en tareas, asegurando un flujo de datos predecible y respetando las reglas de React.

Otro ejemplo de pasaje de props entre componentes

```
1 // $ Card.js
2 export const Card = (props) => {
3   return (
4     <div>
5       <h3>{props.title}</h3>
6       <p>{props.content}</p>
7     </div>
8   );
9 };
```

```
1 // App.js
2 import { Card } from './Card';
3 function App() {
4   return (
5     <div>
6       <Card title="Tarjeta 1" content="Contenido de la tarjeta 1" />
7       <Card title="Tarjeta 2" content="Contenido de la tarjeta 2" />
8     </div>
9   );
10 }
11 export default App;
```

props.children

La propiedad especial **props.children** permite que un componente reciba y renderice elementos anidados dentro de su etiqueta. Esto quiere decir que puedes pasar cualquier tipo de contenido (texto, elementos HTML, otros componentes) dentro de un componente y este lo **renderizará** en la posición de props.children. Esto hace que los componentes sean más dinámicos y reutilizables.

Es útil para crear **contenedores genéricos** que puedan envolver diferentes tipos de contenido.

Una diferencia clave de esta prop es que no hay que pasarla dentro de los **</>** sino que vamos a abrir y cerrar como si fuera una etiqueta html y dentro de ella vamos a pasar el contenido que queremos que se renderice.

```
1 // cajita.js
2 export const Cajita = (props) => {
3   return (
4     <div>
5       {props.children}
6       // Todo lo que vaya a estar entre las etiquetas de apertura y cierre
7       // del componente se va a renderizar acá. Son los hijos de la cajita
8     </div>
9   );
10  };
```



```
1 // App.js
2 import { Cajita } from './Cajita';
3 function App() {
4   return (
5     <div>
6       <Cajita>
7         <h2>Contenido dentro de la caja</ h2>
8         <p>Esto es pasado como children.</ p>
9       </Cajita>
10      <Cajita>
11        <button>Botón dentro de otra caja</ button>
12      </Cajita>
13    </div>
14  );
15 }
16 export default App;
```

Como se ve en la imagen, se podría decir que ‘Cajita’ funciona como una etiqueta de HTML, ya que se abre en la línea 7 y se cierra en la línea 10 (lo mismo sucede en las líneas 11 y 13 respectivamente). Por eso decimos que todo lo que esté dentro de esas ‘etiquetas’ son la propiedad ‘children’. En el primer caso, el h2 y p serían los hijos (children) de la Cajita. En el caso 2 el button sería el hijo (children).

ejemplo

Nos han enviado un código de react que debemos refactorizar. Para ello necesitamos:

- Identificar los elementos que pueden ser componentes
- Crear los componentes necesarios
- Separar el código en archivos (Modularizar)
- Importar los componentes necesarios en el archivo principal
- Utilizar los componentes en el archivo principal



```
1 // $ App.js
2 import React from 'react';
3 function App() {
4   return (
5     <div>
6       <h1>Mi Aplicación React</ h1>
7       <div>
8         <h3>Card 1</h3>
9         <p>Contenido de la Card 1</ p>
10        <button>Aceptar</ button>
11        <button>Cancelar</ button>
12      </div>
13      <div>
14        <h3>Card 2</h3>
15        <p>Contenido de la Card 2</ p>
16        <button>Enviar</ button>
17      </div>
18    </div>
19  );
20 }
21 export default App;
```

```
1 // $ App.js
2 import React from "react";
3 function App() {
4   function Button(props) {
5     return <button>{props.label}</button>;
6   }
7   function Card(props) {
8     return (
9       <div>
10         <h3>{props.title}</h3>
11         {props.children}
12       </div>
13     );
14   }
15   return (
16     <>
17       <h1>Mi Aplicación React</h1>
18       <Card title="Card 1">
19         <p>Contenido de la Card 1</p>
20         <Button label="Aceptar" />
21         <Button label="Cancelar" />
22       </Card>
23       <Card title="Card 2">
24         <p>Contenido de la Card 2</p>
25         <Button label="Enviar" />
26       </Card>
27     </>
28   );
29 }
30 export default App;
```

Así debería quedar el ejercicio final

```
1 // $ Button.js
2 export const Button = (props) => {
3   return (
4     <button>
5       {props.label}
6     </button>
7   );
8 }
```

```
1 // $ Card.js
2 export const Card = (props) => {
3   return (
4     <div>
5       <h3>{props.title}</h3>
6       {props.children}
7     </div>
8   );
9 }
```

```
1 // $ App.js
2 import { Button } from './Button';
3 import { Card } from './Card';
4 function App() {
5   return (
6     <>
7       <h1>Mi Aplicación React</ h1>
8       <Card title="Card 1">
9         <p>Contenido de la Card 1</ p>
10        <Button label="Aceptar" />
11        <Button label="Cancelar" />
12      </Card>
13      <Card title="Card 2">
14        <p>Contenido de la Card 2</ p>
15        <Button label="Enviar" />
16      </Card>
17    </>
18  );
19 }
20 export default App;
```

Las ventajas de hacer esto, por mas que a simple viste parezca que escribimos más código, es que el componente Button y el componente Card, ahora están disponibles para que los utilicemos en cualquier otro archivo sin tener que crearlos nuevamente, solo importándolo ya tendremos los elementos disponibles.



revolución*
digital_