

1- Parte A: Props básicas (Padre -> Hijo).

```

function ChildComponent(props) {
  return <p>Hello bros! my name is {props.name}</p>
}

function App() {
  return <ChildComponent name="Alex1" />;
}

export default App;
  
```

2- Parte B: Pasar múltiples props.

He creado un archivo ChildComponent.jsx, Que devuelve los datos que se crean en App.jsx.

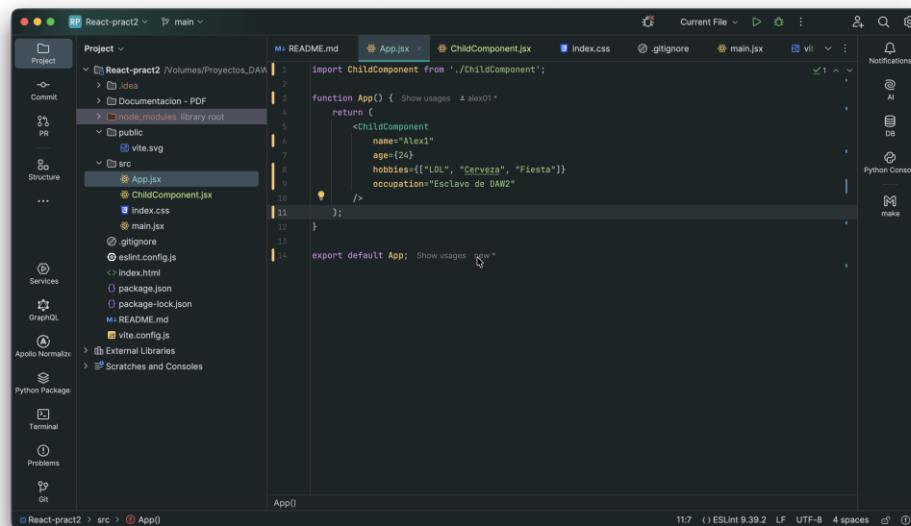
Aquí se puede ver los props, que manda ChildComponent.jsx

```

function ChildComponent(props) {
  return (
    <p>Hello bros! my name is {props.name}</p>
    <p>Tengo {props.age} años</p>
    <p>Soy {props.occupation}</p>
    <p>Mis hobbies: {props.hobbies.join( ' ' )}</p>
  );
}

export default ChildComponent;
  
```

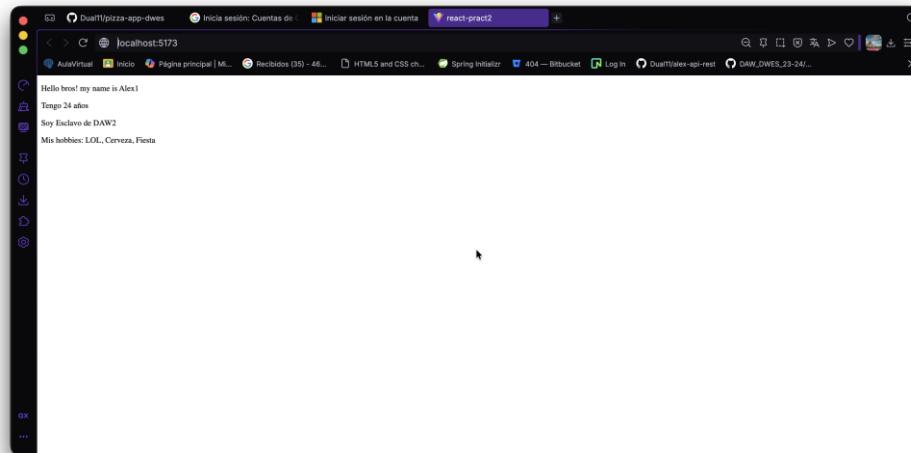
En esta otra img, se puede ver lo que App.jsx, crea con los campos de la anterior img, y en la ultima el resultado.



```
import ChildComponent from './ChildComponent';

function App() {
  return (
    <ChildComponent
      name="Alex1"
      age={24}
      hobbies={['LOL', 'Cerveza', 'Fiesta']}
      occupation="Esclavo de DAW2"
    />
  );
}

export default App;
```



3- Parte C: Pasar funciones como props

En el ChildComponent.jsx, he agregado la llamada del método creado por App.jsx

Luego en App.jsx, llama a ChildComponent.jsx, con el nuevo mensaje creado y lo muestra

```

function greetings(): string {
  return "Hola como va todo"
}

return (
  <ChildComponent
    name="Alex"
    age={24}
    hobbies={['LOL', 'Cerveza', 'Fiesta']}
    occupation="Esclavo de DAW2"
    greetings={greetings}
  />
);

```

En esta imagen, se puede ver cómo es llamada por el método

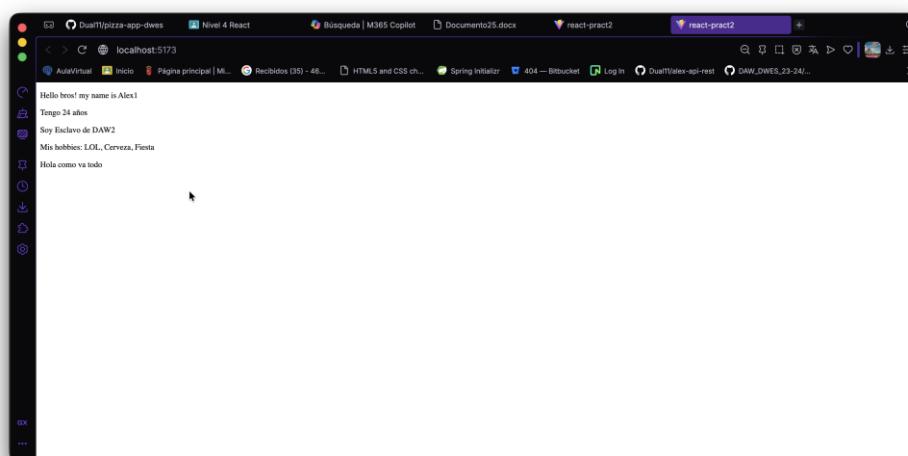
```

function greetings(): string {
  return "Hola como va todo"
}

return (
  <ChildComponent
    name="Alex"
    age={24}
    hobbies={['LOL', 'Cerveza', 'Fiesta']}
    occupation="Esclavo de DAW2"
    greetings={greetings}
  />
);

```

Este es el resultado



4- Parte D: Las props son inmutables

Explicación: Esto pasa porque el hijo no puede modificar lo que recibe el padre, que en este caso es App.jsx, ya que los props son de solo lectura y React los protege.

The screenshot shows a code editor with a dark theme. The file `ChildComponent.jsx` is open, and there is a red error icon next to the line `props.name = "nombre prohibido";`. The code in `ChildComponent.jsx` is as follows:

```

function ChildComponent(props) {
  props.name = "nombre prohibido";

  return (
    <>
      <p>Hello bro! my name is {props.name}</p>
      <p>Iengo {props.age} años</p>
      <p>Soy {props.occupation}</p>
      <p>Mis hobbies: {props.hobbies.join(", ")</p>
    </>
  );
}

export default ChildComponent;
  
```

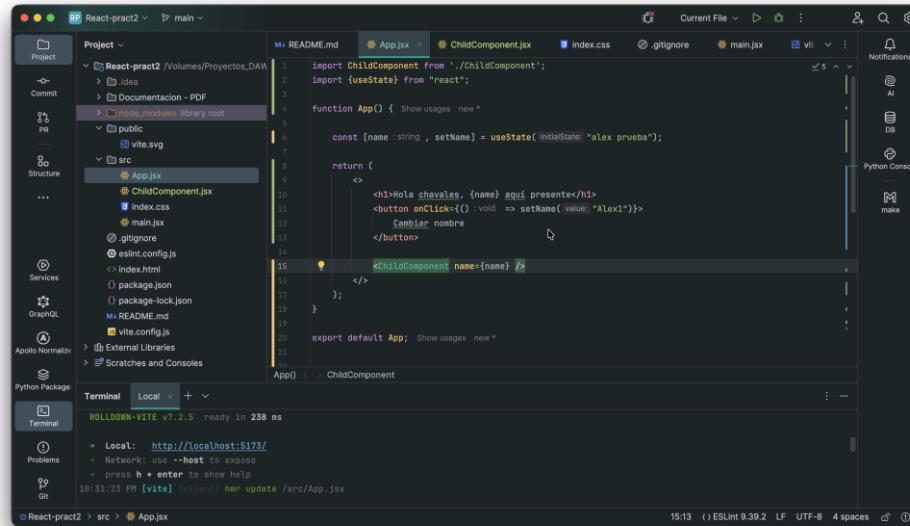
The terminal below shows the command `npm update /src/App.jsx` was run at 10:31:23 PM [vite].

En esta imagen se puede ver el error al código añadido

The screenshot shows the Chrome DevTools Performance tab. A tooltip is displayed over the line `props.name = "nombre prohibido";` in the `ChildComponent.jsx` file, indicating a syntax error. The tooltip text is: "Google Chrome no es tu navegador predeterminado". The right side of the screen shows the Performance panel with various monitoring options like Reloj, Puntos de interrupción, and Alcance.

5- Parte E: Estado con useState

Se ha creado el método en App.jsx, de cambiar el nombre



```

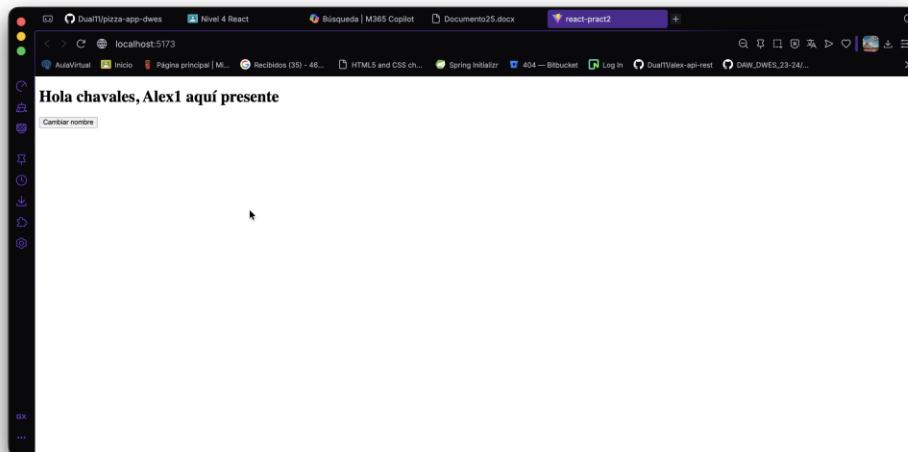
import ChildComponent from './childComponent';
import {useState} from "react";

function App() {
  const [name : string , setName] = useState(initialState: "alex prueba");

  return (
    <>
      <h1>Hola chavales, {name} aquí presento</h1>
      <button onClick={() :void => setName( value: "Alex1")}>
        Cambiar nombre
      </button>
    </>
  );
}

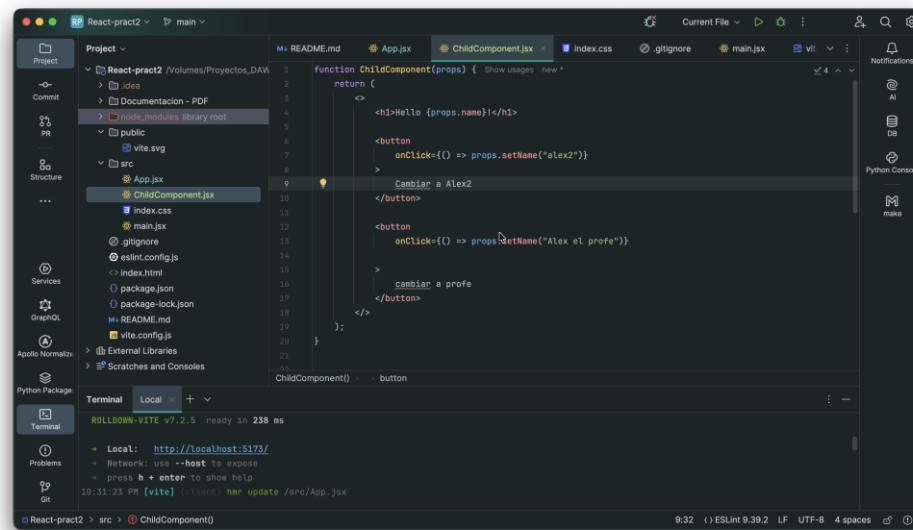
export default App; Show usages new *
  
```

Este es el resultado, se puede ver en el código el nombre que había antes



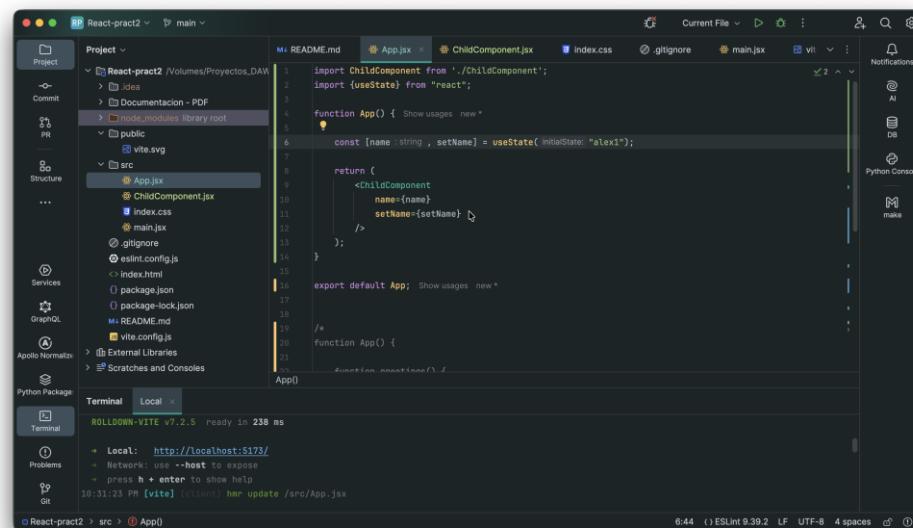
6- Parte F: Pasar estado y setState a un componente hijo

Se han creado los dos botones en ChildComponent.jsx, que luego es llamado por el padre que es App.jsx



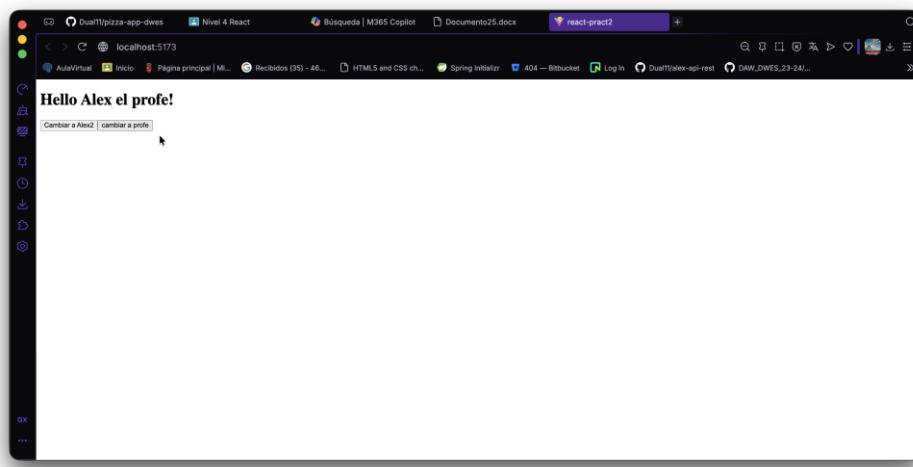
```
function ChildComponent(props) {  
  return (  
    <h1>Hello {props.name}</h1>  
    <button  
      onClick={() => props.setName("alex2")}>  
      Cambiar a Alex2  
    </button>  
    <button  
      onClick={() => props.setName("Alex el profe")}>  
      cambiar a profe  
    </button>  
  );  
}  
  
ROLLODOWN-VITE v7.2.5 ready in 238 ms  
+ Local: http://localhost:5175/  
+ Network: use --host to expose  
+ press h + enter to show help  
10:31:23 PM [vite] (client) HMR update /src/App.jsx
```

En esta imagen, se puede ver la llamada de App.jsx



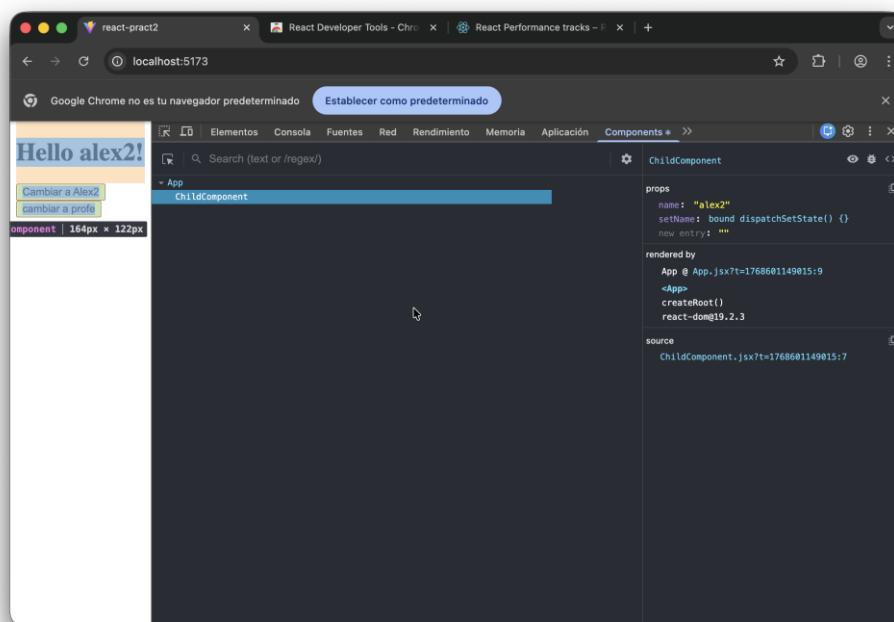
```
import ChildComponent from './ChildComponent';  
import {useState} from "react";  
  
function App() {  
  const [name : string , setName] = useState( initialState: "alex1");  
  
  return (  
    <ChildComponent  
      name={name}  
      setName={setname} >  
  );  
}  
  
export default App; Show usages new *  
  
ROLLODOWN-VITE v7.2.5 ready in 238 ms  
+ Local: http://localhost:5175/  
+ Network: use --host to expose  
+ press h + enter to show help  
10:31:23 PM [vite] (client) HMR update /src/App.jsx
```

Este es el resultado



7- Parte G: Inspeccionar props y state con React DevTools

Primer Ejemplo – alex1



Segundo Ejemplo – Alex el profe

