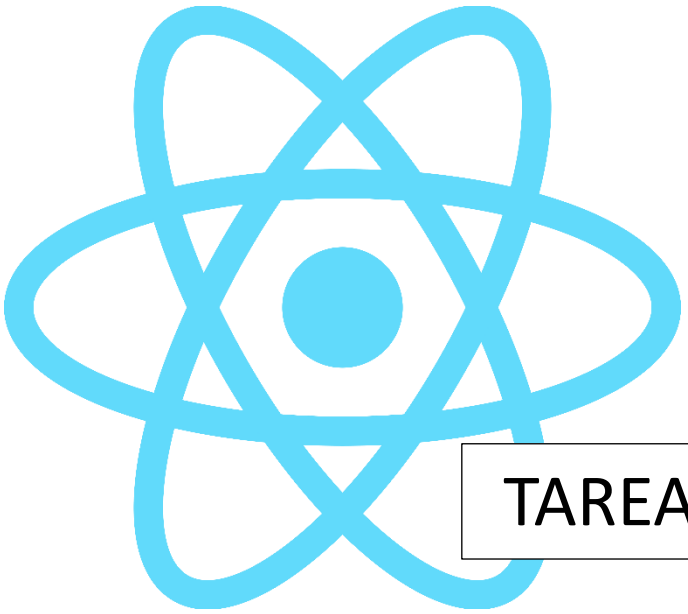


TAREA 4: REACT

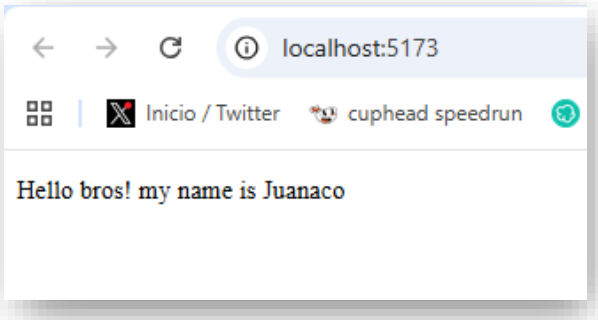
Contenido

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

```jsx

function ParentComponent() {
return <ChildComponent name="Juanaco" />
}

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

export default ParentComponent;
```

"En este apartado, el componente ParentComponent le pasa el valor 'Juanaco' al componente ChildComponent a través de una **prop** llamada name. El hijo recibe este objeto y lo renderiza dinámicamente usando llaves."

Ilustración 1. Props básicas (Padre -> Hijo). Elaboración propia



Resultado en pantalla de las funciones.

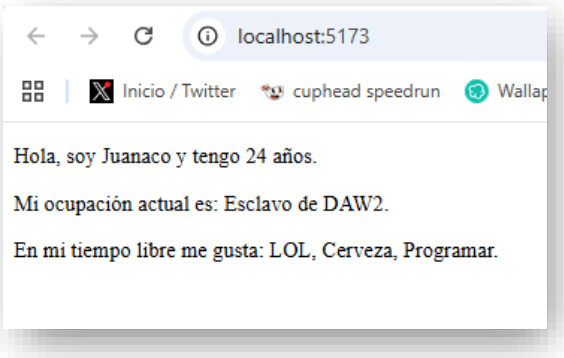
Ilustración 2. Visualización de Resultado. Elaboración propia

Parte B: Pasar múltiples props.

```jsx
function ParentComponent() {
  return (
    <ChildComponent
      name="Juanaco"
      age={24}
      hobbies={["LOL", "Cerveza", "Programar"]}
      occupation="Esclavo de DAW2"
    />
  );
}

function ChildComponent(props) {
  return (
    <div>
      <p>Hola, soy {props.name} y tengo {props.age} años.</p>
      <p>Mi ocupación actual es: {props.occupation}.</p>
      <p>En mi tiempo libre me gusta: {props.hobbies.join(", ")}.</p>
    </div>
  );
}

export default ParentComponent;
```

En este apartado, el componente padre envía diversos tipos de datos al hijo: un **texto** (nombre), un **número** (edad), un **array** (hobbies) y otra cadena para la ocupación. El hijo recibe todas estas **props** y las organiza dentro de etiquetas HTML para mostrarlas de forma estructurada en la interfaz.

Ilustración 3. Multiples props. Elaboración propia



Resultado en pantalla de las funciones.

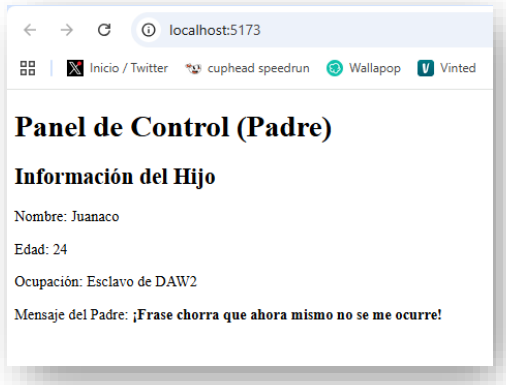
Ilustración 4. Visualización del resultado. Elaboración

Parte C: Pasar funciones como props.

```jsx
function ParentComponent() {
  // Función de la Parte C
  function greetings() {
    return "¡Frase chorra que ahora mismo no se me ocurre!";
  }

  return (
    <div>
      <h1>Panel de Control (Padre)</h1>
      {/* Pasamos todas las props requeridas por los ejercicios */}
      <ChildComponent
        name="Juanaco"
        age={24}
        occupation="Esclavo de DAW2"
        hobbies={["LOL", "Cerveza"]}
        greetings={greetings}
      />
    </div>
  );
}

function ChildComponent(props) {
  return (
    <div>
      <h2>Información del Hijo</h2>
      {/* Parte A y B: Mostrando múltiples props */}
      <p>Nombre: {props.name}</p>
      <p>Edad: {props.age}</p>
      <p>Ocupación: {props.occupation}</p>

      {/* Parte C: Ejecutando la función recibida */}
      <p>Mensaje del Padre: <strong>{props.greetings()}</strong></p>
    </div>
  );
}

export default ParentComponent;
```

En este apartado he juntado todas las props (nombre, edad, trabajo y hobbies) y la función de saludo en un solo código. Así el padre le pasa toda la información de golpe y el hijo la organiza para que se vea bien en la web".

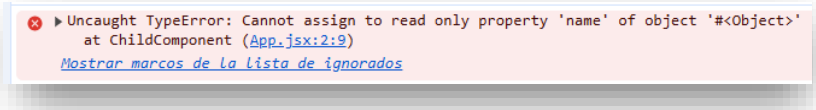
Ilustración 5. Pasar funciones como props. Elaboración propia



Resultado en pantalla de las funciones.

Ilustración 6. Visualización del resultado. Elaboración propia

Parte D: Las props son inmutables



El error ocurre porque en React las **props son de solo lectura** y no se pueden modificar dentro del hijo. Al intentar cambiar el nombre a 'Pablo', React bloquea la aplicación para asegurar que solo el padre controle los datos.

Ilustración 7. Observar el error. Elaboración propia

Parte E: Estado con useState

```jsx
import { useState } from "react";

function ParentComponent() {
  // 1. Crea un estado name con valor inicial 'Juanaco'
  const [name, setName] = useState("Juanaco");

  return (
    <>
      {/* 2. Muestra el nombre en pantalla */}
      <h1>Hola chavales {name}</h1>

      {/* 3. Botón que cambia el nombre a 'Juanky' */}
      <button onClick={() => setName("Juanky")}>
        Cambiar Juanky
      </button>

      {/* 4. Segundo botón que cambia el nombre a otro diferente (Iker) */}
      <button onClick={() => setName("Iker")}>
        Cambiar a Iker
      </button>
    </>
  );
}

export default ParentComponent;
```

En este apartado, el padre le envía una **función** al hijo como si fuera una prop común. El componente hijo recibe esta función y la **ejecuta** dentro de su propio código para mostrar el mensaje de saludo que el padre ha definido.

Ilustración 8. Estado con useState. Elaboración propia

Resultado en pantalla de las funciones, si le damos a Cambiar Juanky se cambiara el nombre.

Ilustración 9. Visualización del resultado. Elaboración propia

Parte G: Inspeccionar props y state con React DevTools

1. Instala la extensión React Developer Tools.

2. Abre tu aplicación en localhost y abre las herramientas de desarrollo.

3. En la barra de pestañas, localiza 'Components' y selecciona ParentComponent y ChildComponent.

4. Haz una captura donde se vean claramente las props y el state.

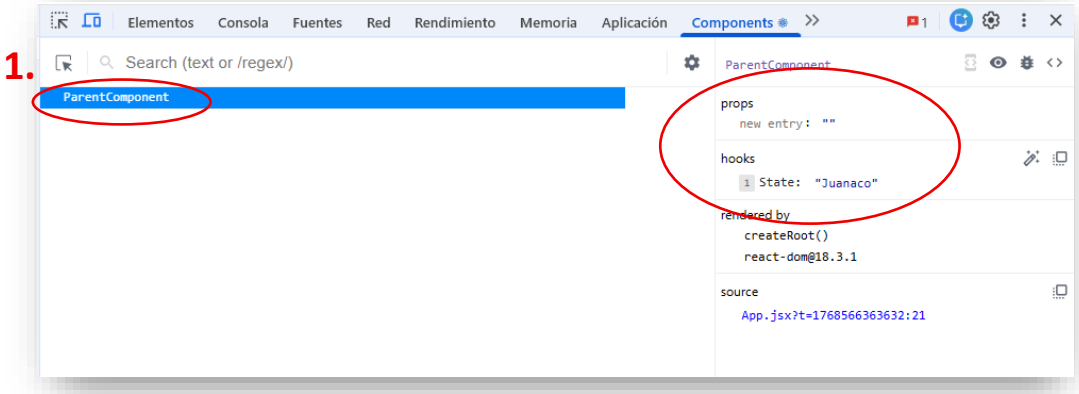


Ilustración 10. Visualización del props y State. Elaboración propia

5. Pulsa el botón de cambio de nombre y captura el state actualizado.

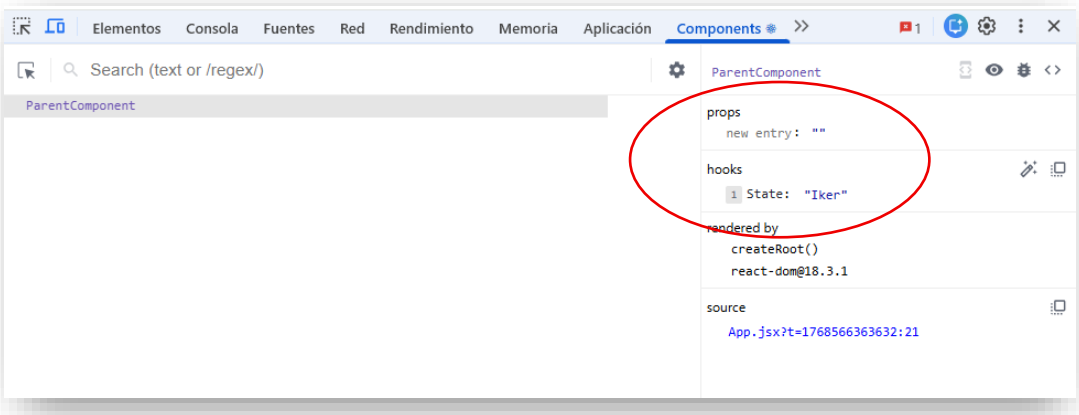


Ilustración 11. Visualización del cambio de props y State. Elaboración propia