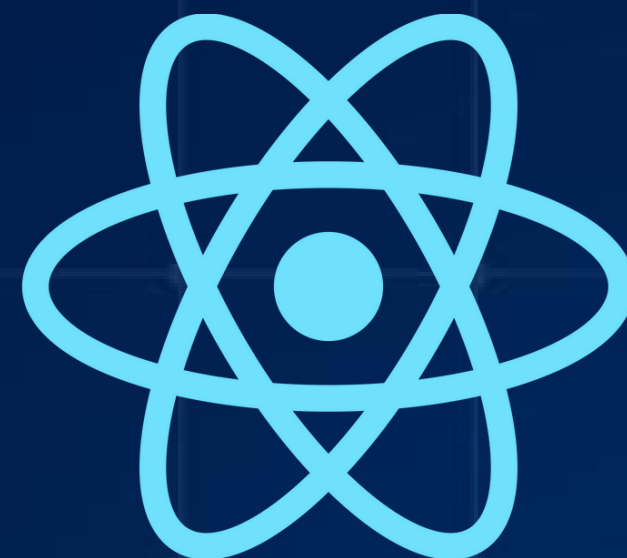


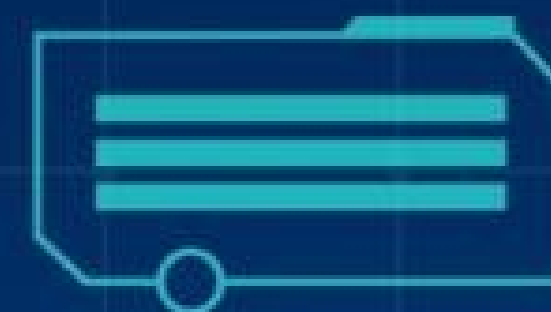
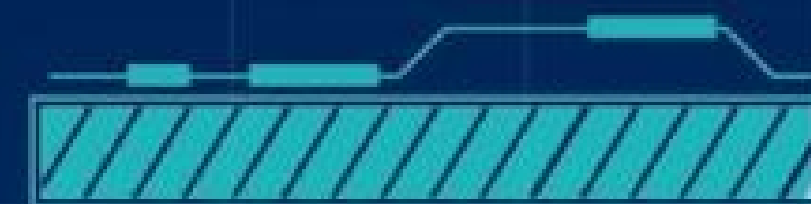
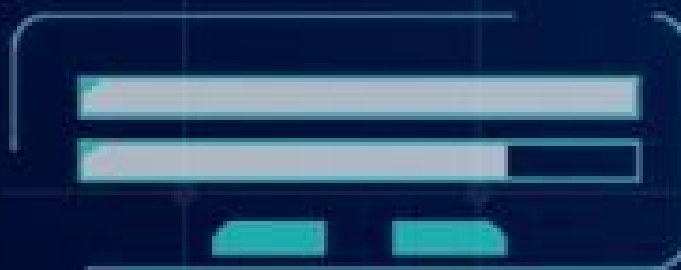


Escuela
Nacional de
Estudios
Superiores

IECAGto[®]
Instituto Estatal de Capacitación



DESARROLLO WEB CON REACT



Temario

Fundamentos de desarrollo web

Javascript

Aplicaciones con React

Hooks y navegabilidad

Consumo de Web APIs

Herramientas en la nube

Sitios estáticos y SSR

Aplicaciones en tiempo real




Consumo de Web APIs

useReducer

useReducer es un Hook que nos permite manipular un estados más complejos de una aplicación o componente de forma predecible.

Su funcionamiento es equivalente a la función **reduce** implementada en el manejo de arreglos en JavaScript, recibiendo un valor inicial y obteniendo un nuevo valor a partir de la combinación de dos valores.

A code editor window with a dark background and light-colored text. It shows a JavaScript code snippet using the useReducer hook. The code is numbered from 1 to 12 on the left side. The code defines a state array [sum, increment] and a reducer function that takes state and type as arguments. The reducer function uses a switch statement to handle different types: 'by_one' increments the state by 1, 'by_two' increments by 2, and the default case returns the state unchanged. The initial state is 0.

```
1
2  const [sum, increment] = useReducer((state, type) => {
3    switch (type) {
4      case "by_one":
5        return state + 1;
6      case "by_two":
7        return state + 2;
8      default:
9        return state;
10   }
11 }, 0);
12
```

useReducer

El hook useReducer puede recibir tres parámetros:

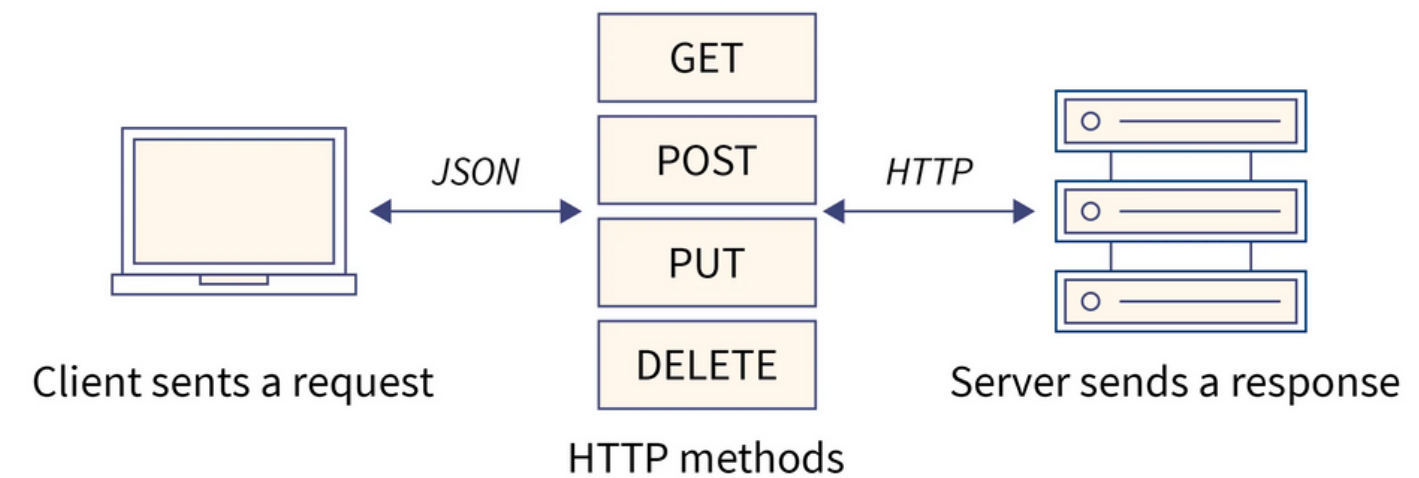
- **reducer:** La función reductora que debe retornar el estado inicial. Debe ser pura, debe tomar el estado y la acción como argumentos, y debe devolver el siguiente estado. El estado y la acción pueden ser de cualquier tipo.
- **initialArg:** El valor a partir del cual se calcula el estado inicial. Puede ser un valor de cualquier tipo. Cómo se calcula el estado inicial depende del siguiente argumento init (opcional).
- **init:** La función inicializadora que especifica cómo se calcula el estado inicial. Si no se especifica, el estado inicial se establece en initialArg. En caso contrario, el estado inicial es el resultado de llamar a init(initialArg)

```
useReducer(reducer, initialArg, init?)
```

Axios

Axios es una biblioteca para el manejo de peticiones HTTP que incorpora funcionalidades más complejas en el manejo de conexiones que la función **fetch()**, añadiendo:

- Conversión automática de objetos JSON.
- Interceptores HTTP.
- Manejo de timeouts.
- Obtener progreso de descargas



AXIOS

Axios

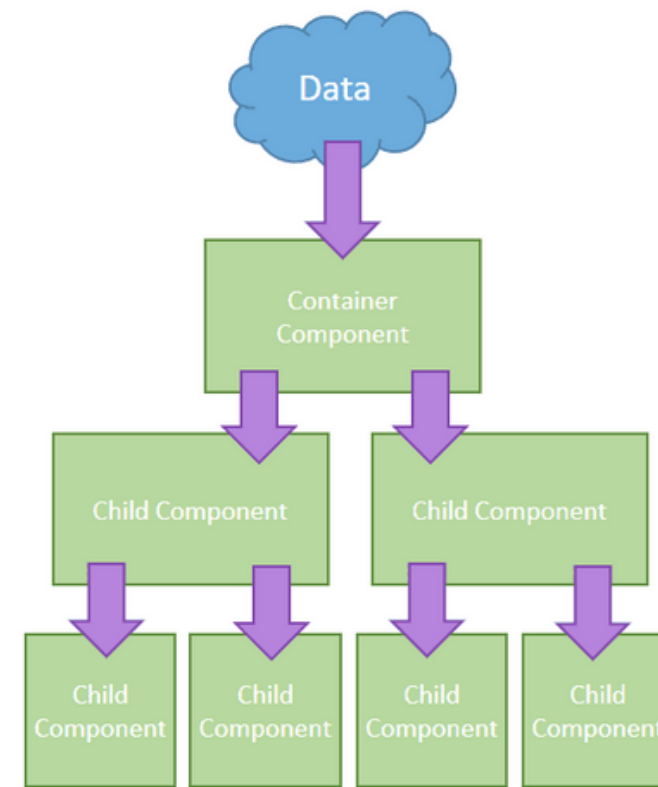
```
// Enviar una petición POST
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

Axios permite realizar peticiones HTTP a servidores remotos utilizando los diversos métodos HTTP (**GET, PUT, PATCH, POST, DELETE**) y obtener respuestas serializadas a partir de estos.

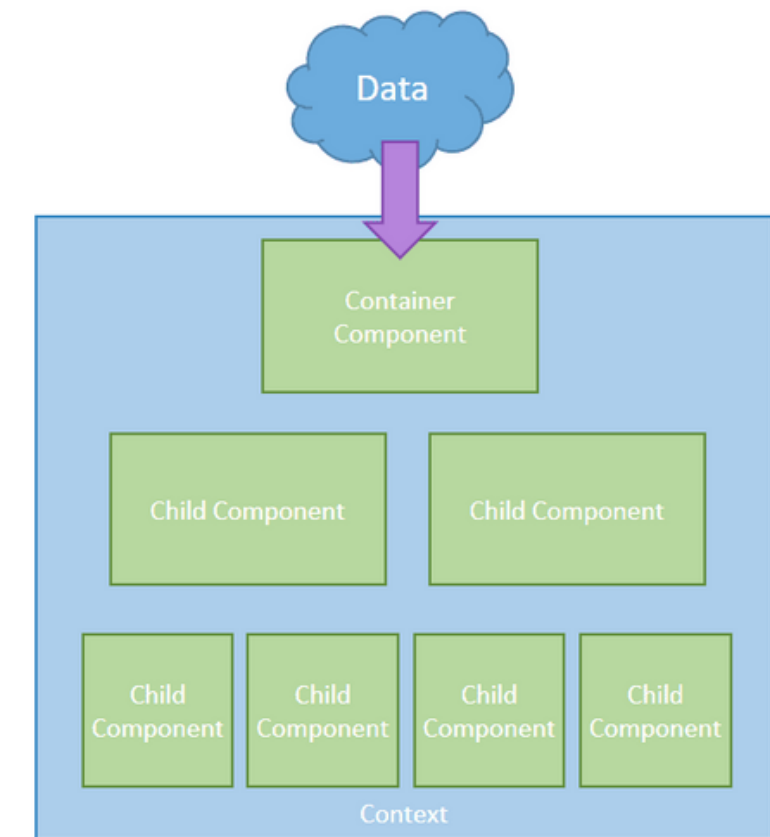
Es ideal para comunicarse con APIs externas y mostrar información del lado del front-end.

React Context

React Context es una característica de React que proporciona una forma de pasar datos a través de la jerarquía de componentes sin tener que pasar explícitamente las props a través de cada nivel de componente.



prop drilling



context API

React Context

El contexto de React está compuesto por tres partes principales:

1. **Proveedor (Provider):** Es un componente de React que proporciona los datos que se compartirán a través del contexto. El Proveedor envuelve a los componentes que necesitan acceder a los datos y les ofrece esos datos a través de su valor de contexto.
2. **Consumidor (Consumer):** Es un componente de React que consume los datos proporcionados por el Proveedor. Los componentes Consumidores se colocan en la jerarquía de componentes donde necesitan acceder a los datos del contexto y reciben automáticamente los datos proporcionados por el Proveedor más cercano en la jerarquía.
3. **Contexto (Context):** Es el objeto que se crea utilizando la función `createContext` de React. Representa el contexto en sí mismo y define el valor inicial que se proporcionará a los componentes Consumidores si no hay un Proveedor que proporcione datos.

React Context

```
1 import React, { createContext, useContext, useState } from "react";
2
3 // Crear el contexto
4 const MyContext = createContext();
5
6 // Crear el hook personalizado para acceder al contexto
7 const useMyContext = () => useContext(MyContext);
8
9 // Componente Proveedor
10 const MyContextProvider = ({ children }) => {
11   const [count, setCount] = useState(0);
12
13   const increment = () => {
14     setCount(count + 1);
15   };
16
17   const decrement = () => {
18     setCount(count - 1);
19   };
20
21   return (
22     <MyContext.Provider value={{ count, increment, decrement }}>
23       {children}
```

```
26   </MyContext.Provider>
27 );
28
29 // Componente Consumidor
30 const Counter = () => {
31   const { count, increment, decrement } = useMyContext();
32
33   return (
34     <div>
35       <p>Contador: {count}</p>
36       <button onClick={increment}>Incrementar</button>
37       <button onClick={decrement}>Decrementar</button>
38     </div>
39   );
40 };
41
42 // Componente principal
43 const App = () => {
44   return (
45     <MyContextProvider>
46       <Counter />
47     </MyContextProvider>
48   );
49 };
50
51 export default App;
52
53 // TODO: Hacer el ejercicio de Tasklist
```



Escuela
Nacional de
Estudios
Superiores

IECAGto[®]
Instituto Estatal de Capacitación



¡GRACIAS!

