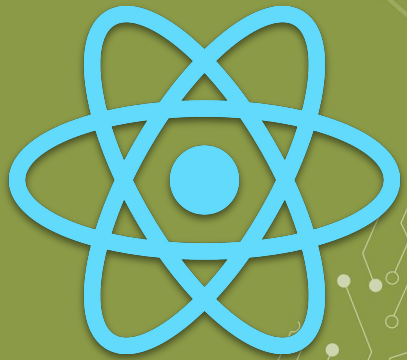
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

Estados



Pregunta Disparadora

¿Cómo pasamos de una web estática como la que venimos haciendo, a algo dinámico?

1

Event Handlers

Agregando interactividad

Eventos

El navegador está constantemente “escuchando” eventos que ocurran. Por ejemplo: hacer un click, hacer scroll, escribir en formulario, mover el mouse, etc. Cualquier acción que hagamos sobre el navegador, disparará un evento.

Entonces, esto es muy útil a la hora de construir aplicaciones dinámicas. Para por ejemplo tener comportamientos al estilo de que cuando el usuario haga click en el boton “⏸️”, pausemos la canción

¿Cómo definimos eventos?

Para poder responder ante un evento que fue disparado por la acción de un usuario.

Tenemos que:

1. Declarar una función que será la que se ejecutará cuando el evento se dispare
2. Pasar esta función como prop, al elemento al cual queremos agregar el evento

Eventos

Definimos la función handleClick y luego lo pasamos por props al button. handleClick es un manejador de eventos.
La cual será ejecutada cuando el evento onClick del botón se dispare

```
export default function Button() {
  function handleClick() {
    alert('Hola! Estoy aprendiendo a manejar enventos');
  }

  return <button onClick={handleClick}>Soy un boton</button>;
}
```

Importante

- La prop de los eventos es en camelCase
- Tenemos que pasar la referencia de la función, no ejecutarla.

Ejercicio

Eventos

2

Estados

La memoria de nuestros componentes - useState

useState

Como resultado de la interacción, nuestros componentes van a tener que cambiar lo que muestran en pantalla. Por ejemplo, Clickeando “agregar” en un ecommerce, debería sumar un producto a nuestro carrito de compras.
Para eso nuestro componente necesita “recordar” cosas, por ejemplo, la cantidad de productos que tenemos en el carrito.
En react a esta “memoria” que tienen nuestros componentes se la llama **estados**

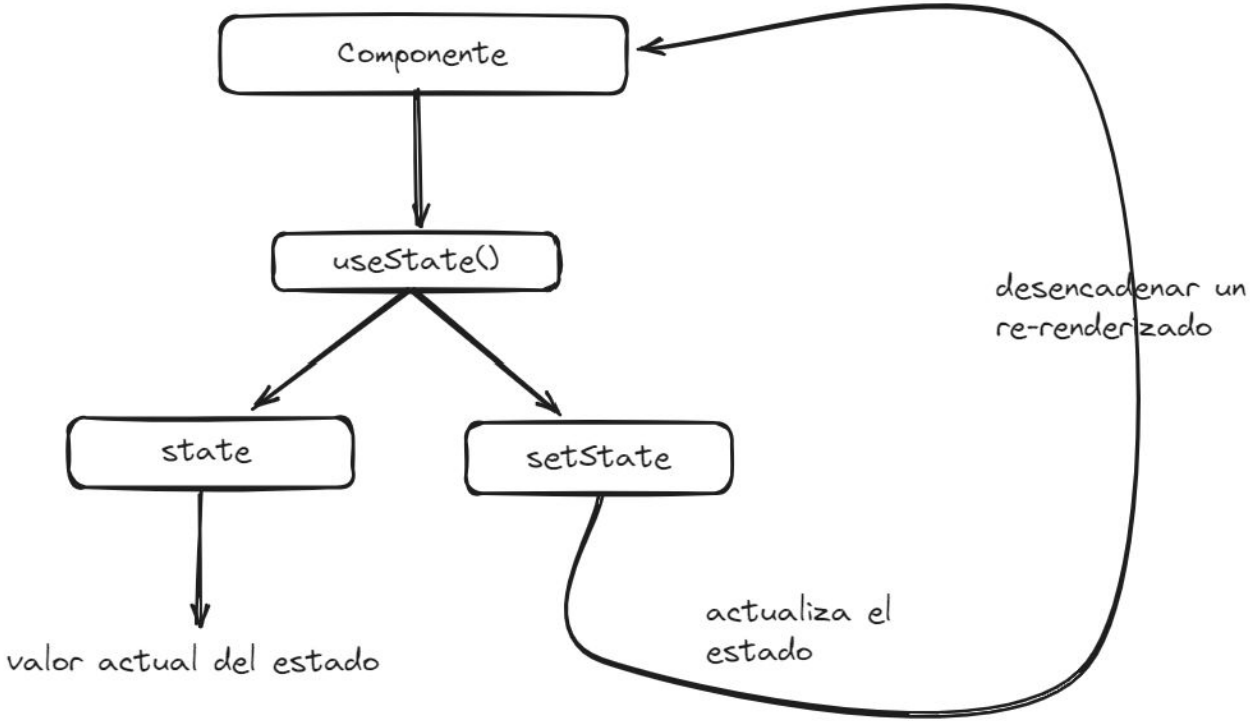
```
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <p>{count}</p>
      <button onClick={handleClick}>+</button>
    </div>
  );
}
```

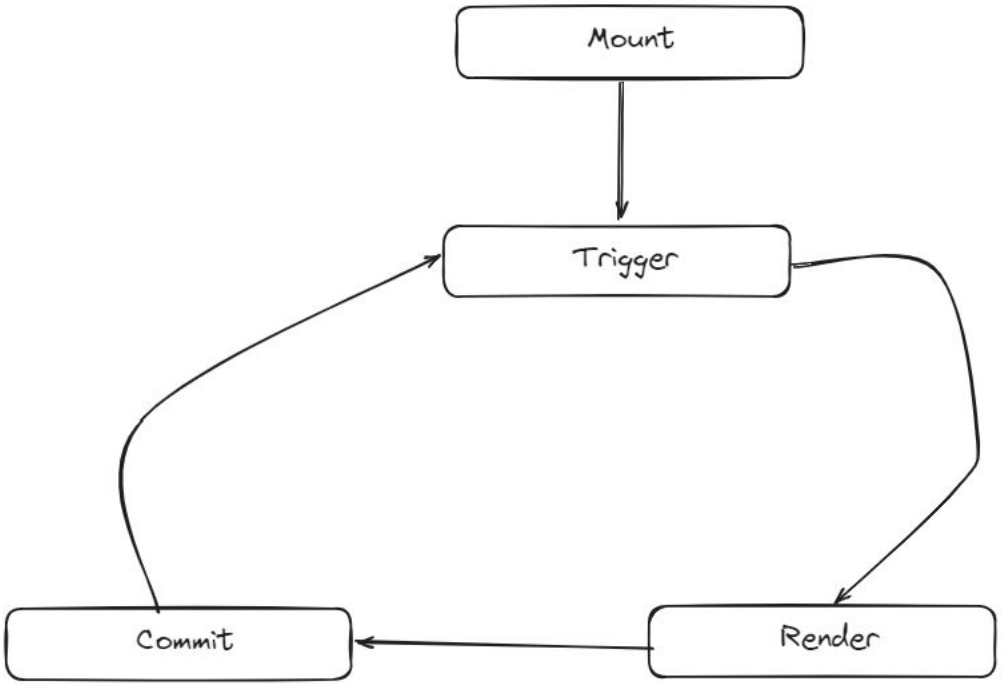
Anatomía de useState



Ejercicio

useState

React Loop



React Loop

1. **Mount:** cuando se renderiza el componente por primera vez. No hay nada que comparar, entonces React crea todos los nodos del DOM desde cero, y los inyecta en la página
2. **Trigger:** sucede alguna interacción, que invoca una función setter (ej. setCount), diciendole a React el valor nuevo del estado
3. **Render:** Al cambiar un estado, necesitamos generar una nueva UI. React invocará nuevamente nuestro componente produciendo un nuevo grupo de elementos y así detectar que necesita actualizar en el DOM
4. **Commit:** Si necesitamos actualizar el DOM, React va a efectuar las modificaciones necesarias. Y luego quedará esperando por nuevas interacciones.

useState

- Se puede tener más de un estado dentro de nuestro componente

- El estado está aislado, privado y solo pertenece a cada **instancia** del componente.

  Es decir, si lo renderizas 2 veces, cada copia tendrá su propio estado o “memoria”

3
Formularios

Data Binding

Cuando construimos aplicaciones, vamos a querer enlazar un estado con un input de un formulario. Por ejemplo, el campo “username”, tiene que estar enlazado con el valor del estado.
Esto lo hacemos utilizando la prop value y onChange.
Que pasa si no realizamos data binding? React, no va a saber que valor tiene el input.
Esto aplica tanto para inputs, como cualquier otro form control (select, checkbox, textarea, etc)

```
export default function Forms() {
  const [username, setUsername] = useState('ivan');

  return (
    <>
      <form>
        <label htmlFor="search-input">Username:</label>
        <input
          type="text"
          id="search-input"
          value={username}
          onChange={(event) => {
            setUsername(event.target.value);
          }}
        />
      </form>
      <p>Tu usuario es: {username}</p>
    </>
  );
}
```

Ejercicio

Formularios

Recursos (anexo)

En stackblitz, quedan ejemplos con el resto de form controls

4

Estados complejos

Estados complejos

Hasta ahora, solo guardamos cosas simples en nuestros estados como números, strings y booleanos.
Pero habrá momentos donde querramos guardar objetos y arrays dentro de un estado.
La verdad es que no hay mucho drama sigue funcionando igual, lo único a tener en cuenta es que los cambios entre estados tienen que ser **inmutables**
Por lo tanto si estamos manejando arrays, al momento de actualizar el estado tenemos que pasar un nuevo array, no mutar el ya existente

```
const [colors, setColors] = useState(['#008000', '#f0f000']);

const colorsJoined = colors.join();
const backgroundImage = `linear-gradient(${colorsJoined})`;

return (
  <div>
    <button
      onClick={() => {
        if (colors.length < 5) {
          const newColors = [...colors];
          newColors.push('#f0f000');
          setColors(newColors);
        }
      }}
    >
      Agregar
    </button>
```

Nunca muten un estado

1. Creen un nuevo array u objeto
2. Modifiquen el array u objeto recien creado
3. Actualicen con el array u objeto recien creado y modificado

Ejercicio

Estados complejos

5

Lifting State up

Una de las estrategias más sencillas para manejar el estado de nuestra aplicación

Lifting State Up