React

Hooks

Enrique Barra Arias
Álvaro Alonso González

Hooks

- Info: https://reactjs.org/docs/hooks-intro.html
- Los Hooks son funciones que te permiten “enganchar” el estado de React y el ciclo de vida **desde componentes de función**
- Los Hooks nos permiten actualizar el estado de modo ordenado y hacer lógica cuando se producen cambios en el ciclo de vida de un componente
  - Ciclo de vida de un componente: montado, actualización y desmontado
- React proporciona algunos Hooks incorporados y también podemos escribir nuestros propios Hooks
- Los más utilizados son los Hooks de estado y efecto
  - useState y useEffect
  - Hay que importarlos de “react”

Hook de estado - useState

- useState devuelve un par: el valor de estado actual y una función que le permite actualizarlo
- Lo igualamos a un array con dos constantes usando Array Destructuring (https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)
- El argumento que recibe useState es el estado inicial (usado para el primer render)

```
export default function Contador() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Contador.js

Hook de estado - useState

- Se pueden utilizar múltiples variables de estado

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

export default function Car() {
  const [brand, setBrand] = useState("Ford");
  const [model, setModel] = useState("Mustang");
  const [year, setYear] = useState("1964");
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My {brand}</h1>
      <p>
        It is a {color} {model} from {year}.
      </p>
    </>
  )
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Car.js
https://github.com/REACT-UPM/ejemplos_simples/tree/main/src/props_state

Hook de estado - useState

- También podría ser el estado en un único objeto “grande”
- Pero para modificarlo tengo que tener cuidado si solo quiero modificar un atributo. Conviene usar el spread operator: https://www.w3schools.com/react/react_usestate.asp

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

export default function Car2() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
    </>
  )
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Car2.js
https://github.com/REACT-UPM/ejemplos_simples/tree/main/src/props_state2

useState - cuando depende del valor anterior

- Si el valor que vamos a actualizar depende del estado anterior es mejor hacerlo con un callback

```jsx
import React, { useState, useEffect } from 'react';

//mejora a contador1 en que el setCount usa el valor anterior
export default function Contador3() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count => count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Contador3.js

Hook de efecto - useEffect

- useEffect agrega la capacidad de realizar “efectos secundarios” desde un componente de función

```jsx
import React, { useState, useEffect } from 'react';

export default function Contador4() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Contador4.js

Hook de efecto - useEffect

- Los efectos se declaran dentro del componente para que tengan acceso a sus props y estado
- React ejecuta los efectos después de cada renderizado
- Si queremos que el hook useEffect se ejecute solo cuando algo concreto del estado cambia, se lo pasamos en un array como segundo parámetro

```
useEffect(() => {
  // Actualiza el título del documento usando la Browser API
  document.title = `You clicked ${count} times`;
}, [count]);
```

Hook de efecto - useEffect - saneamiento o limpieza

- Normalmente nos suscribimos a cosas al montar el componente y nos debemos desuscribir al desmontar

```
useEffect(() => {
    API.subscribe()
    return function cleanup() {
        API.unsubscribe()
    }
})
```

```
useEffect(() => {
  const interval = setInterval(() => {
    console.log('This will run every second!');
  }, 1000);
  return () => clearInterval(interval);
}, []);
```

Hook de efecto - useEffect - recapitulando

```
useEffect(() => {
  // Side-effect...
  return function cleanup() {
    // Side-effect cleanup...
  };
}, [dependencies]);
```

Programa completo

```
export default function Contador5() {
  const [count, setCount] = useState(0);
  const [tick, setTick] = useState(0);

  useEffect(() => {
    const interval = setInterval(()=>{
      setTick(tick => tick+1);
    },1000);
    return () => clearInterval(interval);
  },[]);

  return (<div>
      <div>¿Eres más rápido que un setInterval?</div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
      <p>Han pasado {tick} ticks</p>
    </div>);
}
```

https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/Contador5.js

Otros Hooks

- useContext
- useReducer
- useRef
- useCallback
- useMemo
- useImperativeHandle
- useLayoutEffect
- useDebugValue

- Custom Hooks:
  - https://es.reactjs.org/docs/hooks-custom.html

- + Hooks que nos proporcionan otras librerías (por ejemplo el router con useParam o useHistory)