

Base

React



React, cambio de chip

Uno de los errores que se suelen cometer cuando se empieza a desarrollar con React, es aplicar mentalidad Java, .net o Angular, esto no funciona.

Los componentes de React se basan en funciones.

Las actualizaciones en React son asíncronas.

El estado en React es inmutable

¿Por que? Se busca desarrollar aplicaciones robustas y con buen rendimiento

Esto supone aprender algo nuevo y un cambio de forma de pensar

Funcional

Con la introducción de Hooks en la versión 16.8.0 de React, se ha pasado a desarrollar con una aproximación funcional.

Los componentes de tipo clase se han quedado como Legacy

Podemos tener estado en componentes funcionales vía Hooks

Olvidate de la herencia

A la hora de reusar funcionalidad se prima la composición

Es un cambio de paradigma para los que venimos de tecnologías como .net, java...

<https://en.reactjs.org/docs/composition-vs-inheritance.html>

Ejemplo Componente

Ejemplo de un componente funcional.

```
import React from "react";

export const MyComponent = props => {
  const [myName, setMyName] = React.useState('John');

  return (
    <>
      <h4>{myName}</h4>
      <input
        value={myName}
        onChange={(e) => setMyName(e.target.value)}
      />
    </>
  );
};
```

Un componente es una función, que se ejecuta y destruye

Si se crea y destruye... ¿Cómo tenemos persistencia? ¿Cómo emulamos las variables miembros de clases? Con una genialidad del equipo de React, los Hooks

Un componente de tipo función devuelve un trozo de JSX

Inmutabilidad

Nunca modificamos un objeto o estructura, si necesitamos hacer una modificación, creamos un objeto nuevo

Un objeto se puede usar en varios sitios, si lo mutamos, estamos cambiando sin avisar ¿ Os imagináis que mutáis por error un campo descuento y que afecta a toda la aplicación?

Si muto un objeto de datos asociado a un componente ¿ Cómo se que tengo que repintar el componente? ¿Comparando una a una sus propiedades y subpropiedades?

Si mis objetos son inmutables mi aplicación es predecible, y para saber si ha habido algún modificación sólo tengo que comparar el puntero de un objeto

Trabajar con inmutabilidad no es fácil, algunas ayudas: immer, deepfreeze

<https://immerjs.github.io/immer/docs/introduction>

<https://stackoverflow.com/questions/34385243/why-is-immutability-so-important-or-needed-in-javascript>

Asincronía

En React podemos pedir que se actualice el estado de nuestro componente, pero es el propio motor de React el que decide cuando aplicar esos cambios.

```
import React from "react";

export const MyComponent = () => {
  const [name, setName] = React.useState('John');

  React.useEffect(() => {
    console.log(name); // John
    setName('Mary');
    console.log(name); // John
  }, [])

  return (
    <h2>Hello {name}</h2>
  )
}
```

Esto de primeras puede añadir complejidad a nuestro desarrollo

La naturaleza de una aplicación real es asíncrona (por ejemplo una llamada a servidor)

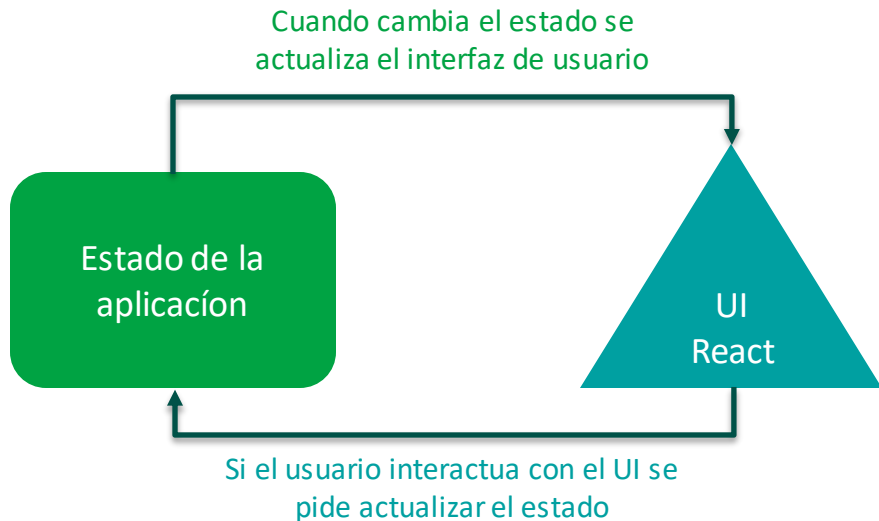
React optimiza el renderizado

Puede llegar a agrupar varios setState en un solo render

Nos obliga a cambiar de mentalidad

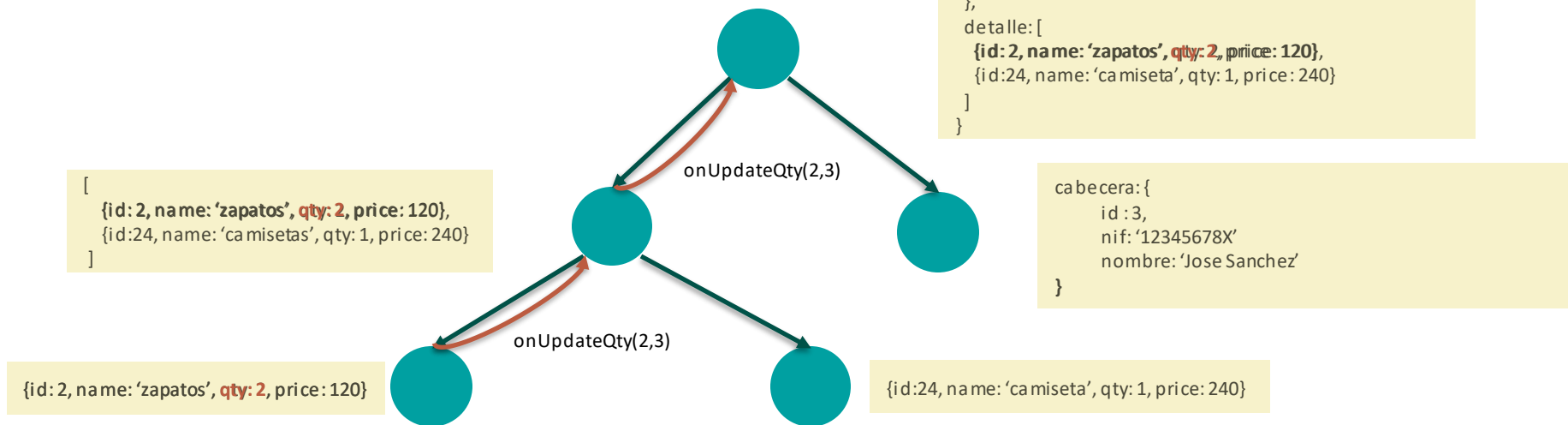
Flujo unidireccional

El paso de datos fluye en un solo sentido.




Flujo unidireccional

Callbacks para actualizar de hijo a padre.



¡ Muchas gracias !



 @lemoncoders



<https://github.com/lemoncode>



 @basefactorteam