

React – Hooks

“React sin usar clases”



■ React – Hooks

- ❑ Introducción a los hooks
- ❑ State hook
- ❑ Effect hook
- ❑ Reglas de los hooks
- ❑ Creando nuestros propios hooks
- ❑ Otros hooks



■ Introducción a los hooks

- ❑ Permiten usar estado y otras features en componentes funcionales
- ❑ A partir de la versión 16.8 de React
- ❑ Son opcionales y 100% retro compatibles, podemos elegir entre usarlos o no
- ❑ Motivación:
 - Un modo más sencillo de reusar lógica entre componentes
 - Componentes complejos son difíciles de entender
 - Clases en javascript son confusas



State hook: useState (I)

Estado en componentes funcionales

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  render() {
    const { count } = this.state;
    return (
      <div>
        <p>You clicked {count} times</p>
        <button
          onClick={() => this.setState({ count: count + 1 })}
        >
          Click me
        </button>
      </div>
    );
  }
}
```

VS

```
function Example() {
  const [count, setCount] = useState(0);

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



■ State hook: useState (II)

- ❑ Estado en componentes funcionales. Se mantiene el valor de count entre renderizados sucesivos.

```
const [count, setCount] = useState(0);
```

- ❑ count: valor del estado
- ❑ setCount: función para establecer el estado
 - setCount(newValue)
 - setCount(value => newValue)
- ❑ useState(0): pasamos un valor inicial al hook



■ Effect hook: useEffect (I)

□ *Side effects* en componentes funcionales

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

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

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



■ State hook: useEffect (II)

□ *Side effects* en componentes funcionales

```
// La función que pasamos se ejecutará tras cada render
useEffect(() => {
  // Cuerpo del side-effect
  ...
  // Podemos devolver una función de cleanup
  // que se ejecutará antes del próximo efecto
  // Cancelar subscripciones, intervals...
  return function cleanup () {}
}, [dependencies]);
// Con el array de dependencias podemos ajustar cuando se ejecuta el effect
// 1. Por defecto siempre
// 2. [] sólo tras el primer render, similar a componentDidMount
// 3. Cuando cambien los valores del array
```



■ Reglas de los hooks

- ❑ Sólo se puede llamar a un hook desde el nivel más alto del componente. **No podemos llamar un hook dentro de:**
 - Bucles
 - Condicionales
 - Funciones anidadas
- ❑ Sólo se puede llamar a un hook desde:
 - Funciones componentes React
 - Otros hooks
- ❑ `eslint-plugin-react-hooks`



■ Creando nuestro propios hooks

- ❑ Extracción de lógica de componentes en funciones reutilizables
- ❑ Custom hook, función javascript que:
 - El nombre empieza por “**use**” (convención)
 - Puede llamar a otros hooks

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });
  return isOnline;
}
```



■ Otros hooks (I)

- `useContext`: conexión a un context React

```
const value = useContext(MyContext);
```

- `useReducer`: alternativa tipo Redux a `useState`, preferible cuando tenemos lógica de estado complicada

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```



■ Otros hooks (II)

□ useCallback: devuelve una función memoizada

```
const memoizedCallback = useCallback(  
  () => { doSomething(a, b); },  
  [a, b],  
);
```

□ useMemo: devuelve un valor memoizado

```
const memoizedValue = useMemo(  
  () => computeExpensiveValue(a, b),  
  [a, b],  
);
```



■ Otros hooks (III)

- ❑ `useRef`: devuelve un objeto ref mutable cuya propiedad *current* es inicializada al argumento pasado como *initialValue*
 - Ref a elementos del DOM
 - Objeto para mantener cualquier valor mutable (como una variable de instancia en una clase)

```
const refContainer = useRef(initialValue);
```

- `useRef` no notifica cuando cambia el valor



React redux hooks

- ❑ `useSelector`: permite extraer datos de un store redux, usando un selector (función que recibe el estado)

```
const result = useSelector(selector);
```

- ❑ `useDispatch`: devuelve una referencia al método *dispatch* del store redux

```
const dispatch = useDispatch();
```

- ❑ `useStore`: devuelve una referencia al store redux que fue pasado al `<Provider />`. Uso muy poco frecuente

```
const store = useStore();
```



■ React router hooks

□ useHistory: acceso al *history*

```
const history = useHistory();
```

□ useLocation: devuelve el objeto location (cambios en la URL)

```
const location = useLocation ();
```

□ useParams: devuelve un objeto con los parámetros de la URL

```
const params = useParams();
```

□ useRouteMatch: devuelve el *match* de la URL con un *path*

```
const match = useRouteMatch(path);
```

