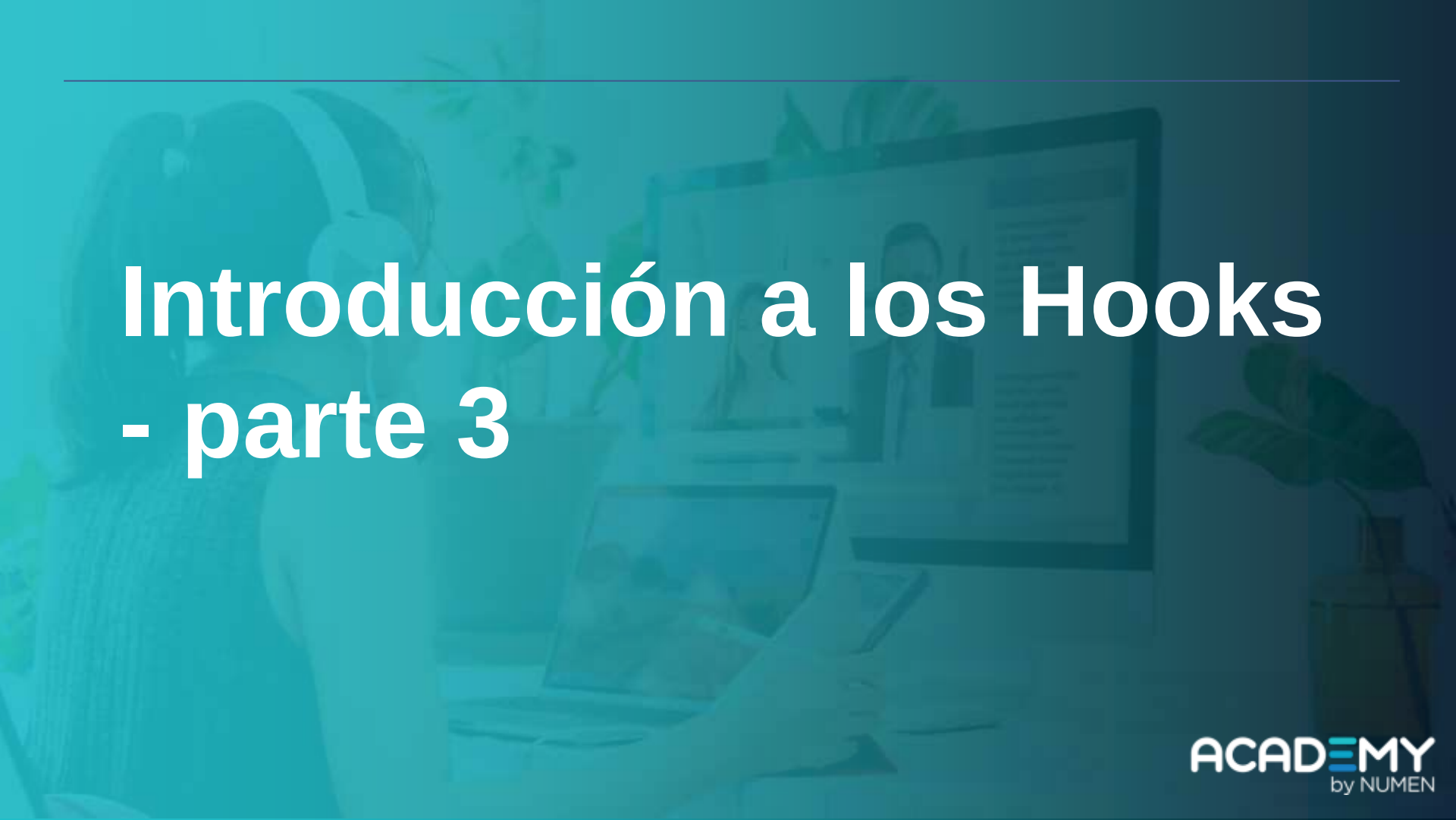
ACADEMY by NUMEN

FULL STACK

# Comenzamos en unos minutos


ACADEMY
by NUMEN

# Introducción a los Hooks - parte 3



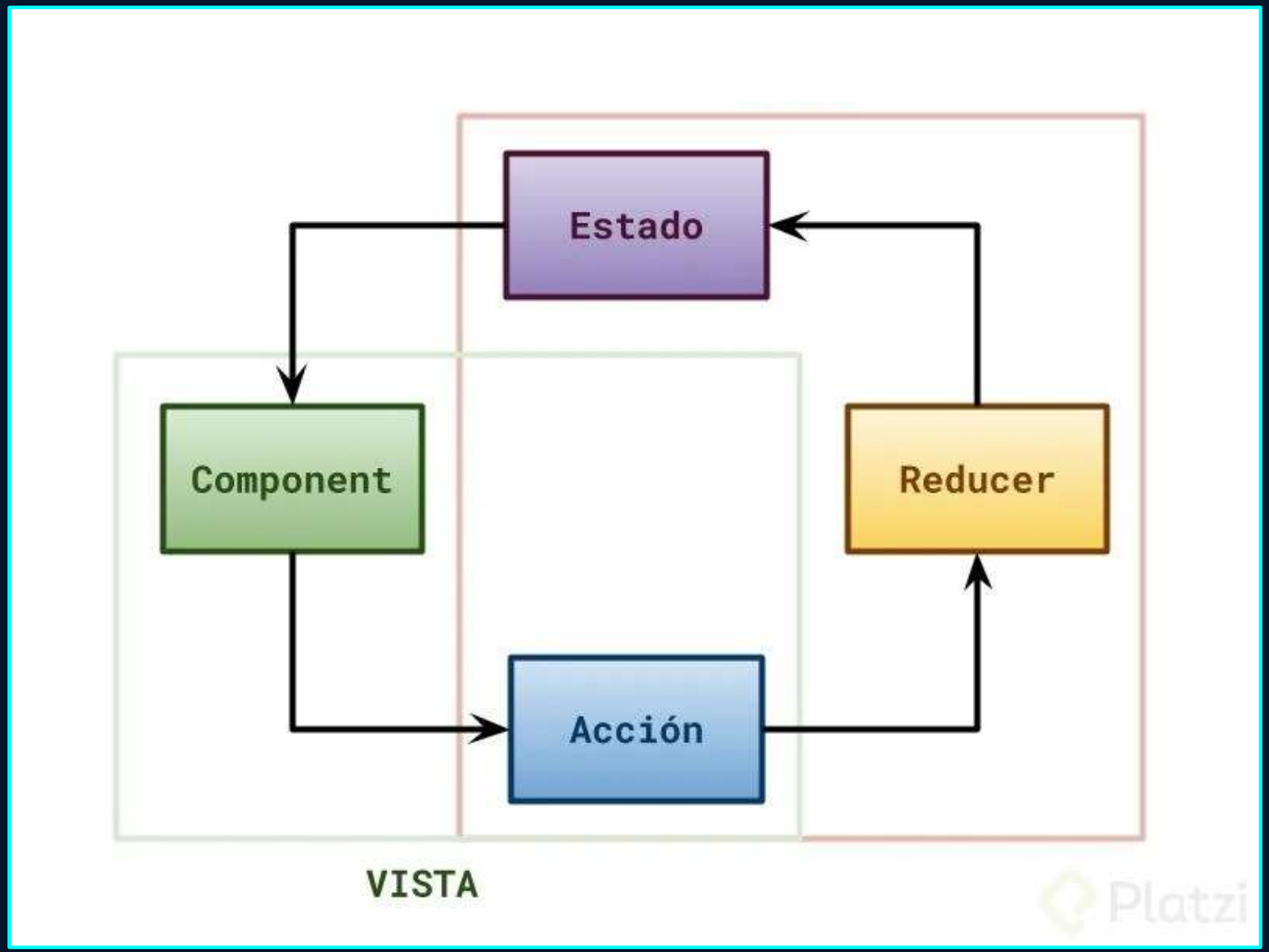
ACADEMY by NUMEN


useReducer

# ¿Qué es useReducer?

🔥 El *useReducer* es un hook que se utiliza para administrar el estado de la aplicación. Es muy similar a *useState* , solo que más complejo.

⚠️ useReducer es ideal para manejar componentes que dependerán de 2 o más estados haciéndolo más escalable y mantenible.




♻️Para programar este hook comenzamos por definir acciones, que para ser ejecutadas es necesario despachar dicha acción por medio de un evento.
La ejecución de estas acciones realizará un cambio en el estado inicial.

# ¿Recuedan nuestro ejemplo de contador?

```
import {useState} from "react"

const Contador = () => {
    const [contador, setContador] = useState(0)

    const sumar = () => setContador(contador + 1);
    const restar = () => setContador(contador - 1);

    return (
        <div style={{textAlign: "center"}}>
            <nav>
                <button onClick={sumar}>+</button>
                <button onClick={restar}>-</button>
            </nav>
            <h3>{contador}</h3>
        </div>
     );
}

export default Contador;
```

Seguramente sí!! 😄

Bien, es hora de empezar lentamente a pasarlo a useReducer 💪💪💪

```jsx
const ContadorReducer = () => {
    // const [contador, setContador] = useState(0)
  const [state, dispatch] = useReducer(reducer,
initialState)

    // const sumar = () => setContador(contador + 1);
  const sumar = () => dispatch({type:"INCREMENTO"});

    // const restar = () => setContador(contador - 1);
  const restar = () => dispatch({type:"DECREMENTO"});

    return (
        <div style={{textAlign: "center"}}>
            <h2>Contador con Reducer</h2>
            <nav>
                <button onClick={sumar}>+</button>
                <button onClick={restar}>-</button>
            </nav>
            <h3>{state.contador}</h3>
        </div>
     );
}
export default ContadorReducer;
```

Instanciamos nuestro useReducer

Importamos useReducer

```jsx
import {useReducer, useState} from "react"

const initialState = {contador: 0};

function reducer(state, action) {
    switch (action.type) {
        case "INCREMENTO":
            return {contador: state.contador + 1};
        case "DECREMENTO":
            return {contador: state.contador - 1};
        default:
            return state;
    }
}
```

Luego establecemos nuestro estado inicial y nuestra función reductora.

Finalmente programamos los despachos.

ACADEMY by NUMEN

# ¿Que es action?

Habrán notado que uno de los parámetros de la función reductora es el **estado** y el otro es uno llamado **action**.

**action** es un objeto que posee 2 propiedades. Una de ellas es la propiedad **type** (que a la vez es un objeto) y sirve para coleccionar todas las acciones que manejaremos con el reductor.

```
const TYPES = {
    incrementar: "INCREMENTAR",
    decrementar: "DECREMENTAR",
}
function reducer(state, action) {
    switch (action.type) {
        case TYPES.incrementar:
            return {contador: state.contador + 1};
        case TYPES.decrementar:
            return {contador: state.contador - 1};
        default:
            return state;
    }
}
```

💡 Para mejorar un poco la estructura podemos separar las acciones en un objeto externo a modo de poder eventualmente pasar las acciones a otra hoja JS para mejorar la escalabilidad del reductor.

```
const sumar = () => dispatch({type:TYPES.incrementar});

const restar = () => dispatch({type:TYPES.decrementar});
```

# Concepto de Payload

🔥 El **payload** es la segunda propiedad que posee el objeto **action** y esta sirve como una especie de variable a la que se le asigna un valor durante la programación del **dispatch**.

⚠️ El **payload** sirve principalmente para programar datos que pretendemos utilizar varias veces a lo largo de la programación del **case** de la función reductora asociado al dispatch al que se le programó ese payload.

```
const TYPES = {
    incrementar: "INCREMENTAR",
    decrementar: "DECREMENTAR",
    incrementar2: "INCREMENTAR_2",
    decrementar2: "DECREMENTAR_2",
    resetear: "RESETEAR"
}
```

⚒ Para aplicar el concepto de payload agregaremos 3 acciones más a nuestro contador.
Dos acciones serán para aumentar el contador de 2 en 2. Y la tercera acción será para resetear el contador.

```
const sumar2 = () => dispatch({type:TYPES.incrementar2, payload: 2});

const sumar = () => dispatch({type:TYPES.incrementar});

const restar = () => dispatch({type:TYPES.decrementar});

const restar2 = () => dispatch({type:TYPES.decrementar2, payload: 2});

const resetear = () => dispatch({type:TYPES.resetear});
```

```
function reducer(state, action) {
    switch (action.type) {
        case TYPES.incrementar:
            return {contador: state.contador + 1};
        case TYPES.incrementar2:
            return {contador: state.contador + action.payload};
        case TYPES.decrementar:
            return {contador: state.contador - 1};
        case TYPES.decrementar2:
            return {contador: state.contador - action.payload};
        case TYPES.resetear:
            return initialState
        default:
            return state;
    }
}
```

```
<nav>
    <button onClick={sumar2}>+2</button>
    <button onClick={sumar}>+</button>
    <button
onClick={resetear}>Resetear</button>
    <button onClick={restar}>-</button>
    <button onClick={restar2}>-2</button>
```

Es hora de programar nuestras nuevas acciones usando el payload que previamente declaramos en los dispatch.

También crearemos 3 botones más en el JSX y le pasaremos sus respectivas funciones.

# Mejorando la escalabilidad de nuestro reductor

Una vez listo nuestro contador con reductor solamente nos queda pensar en su escalabilidad.

Para ello separaremos el objeto **types** en una hoja aparte llamada **actions.js** y también separaremos ambos **initialstate** y **función reductora** en una hoja llamada **reducer.js**

Luego simplemente exportamos esas hojas y las importamos en el componente y listo. En caso de volverse muy grande no comprometerá la estructura y lectura de nuestro componente.

ACADEMY
by NUMEN