



>Xtratego
ACADEMY



Unidad 2

Estado y Ciclo de Vida de
Componentes



>Xtratego
ACADEMY



Unidad 2.1

Estado en React (State)

El **estado** en React es un **objeto** que almacena **datos dinámicos** de un **componente**, permitiendo que el componente responda a cambios en la **interfaz de usuario UI**.

A diferencia de las **props**, que son **inmutables** y se pasan desde el **componente padre**, el **estado** es **mutable** y está gestionado internamente por el **componente**.

En los **componentes** de clase, el estado se inicializa en el constructor y se actualiza mediante el método **setState**.

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  
  
  increment = () => {  
    this.setState({ count: this.state.count + 1 });  
  }  
  
  render() {  
    return (  
      <div>  
        <p>Contador: {this.state.count}</p>  
        <button onClick={this.increment}>Incrementar</button>  
      </div>  
    );  
  }  
}
```

Ejemplo: modulo2/contador-antd

Desde la introducción de **hooks** en **React 16.8**, el estado puede manejarse en componentes funcionales utilizando el hook **useState**

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

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

  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrementar</button>
    </div>
  );
}
```

Ejemplo: modulo2/contador-antd-2

Estado en componentes de clase

useState devuelve un **array** con dos elementos: **el valor actual del estado y una función para actualizarlo**

State Name

State Default Value

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

State Change
Function Name

The diagram illustrates the components of the `useState` hook. It shows the code `const [count, setCount] = useState(0);` with three red arrows pointing to specific parts: one from 'State Name' to `count`, one from 'State Default Value' to `0`, and one from 'State Change Function Name' to `setCount`.



>Xtratego
ACADEMY



Unidad 2.2

Actualización del Estado y
Renderizado

Cuando se actualiza el estado de un componente, **React** vuelve a renderizar dicho componente para reflejar los cambios en la interfaz de usuario UI.

```
this.state.count = this.state.count + 1; // Incorrecto
```

```
this.setState({ count: this.state.count + 1 }); // Correcto
```




>Xtratego
ACADEMY



Unidad 2.3

Ciclo de Vida de los Componentes

El ciclo de vida de un componente en **React** abarca desde su **creación** hasta su **destrucción**.

React proporciona **métodos** para que los desarrolladores **controlen** diferentes **fases del ciclo de vida** en los componentes de clase.

Ocurre cuando las props o el estado cambian

componentDidUpdate(prevProps, prevState): Se ejecuta justo después de una actualización. Ideal para realizar acciones basadas en cambios de estado o props

```
componentDidUpdate(prevProps) {  
  if (this.props.id !== prevProps.id) {  
    this.fetchData(this.props.id);  
  }  
}
```

Ocurre cuando el componente se inserta en el DOM

componentDidMount(): Se ejecuta inmediatamente después de que el componente ha sido montado. Ideal para realizar tareas como llamadas a APIs

```
componentDidMount() {  
  fetch('https://api.example.com/data')  
    .then(response => response.json())  
    .then(data => this.setState({ data }));  
}
```

<https://jsonplaceholder.typicode.com/>

Ocurre cuando el componente es eliminado del DOM

componentWillUnmount(): Se ejecuta justo antes de que el componente sea destruido. Útil para limpiar suscripciones o temporizadores.

```
componentWillUnmount() {  
  clearInterval(this.timer);  
}
```

Ejemplo: modulo2/api-antd-project

En los componentes funcionales, el **hook useEffect** reemplaza la funcionalidad de varios métodos del ciclo de vida

```
useEffect(() => {  
  document.title = `Has hecho clic ${count} veces`;  
}, [count]); // Se ejecuta solo cuando cambia el valor de count
```

useEffect: Se ejecuta después de que el componente se ha renderizado

Se puede **configurar** para que:

- Se ejecute después de cada renderizado.
- Se ejecute solo cuando cambien determinadas variables (**similar a componentDidUpdate**).
- Limpie recursos (**similar a componentWillUnmount**).

```
useEffect(() => {  
  const timer = setInterval(() => {  
    setCount(count + 1);  
  }, 1000);  
  
  return () => clearInterval(timer); // Limpieza similar a componentWillUnmount  
}, []);
```



>Xtratego
ACADEMY



Unidad 2.4

Manejo de Eventos en React

React utiliza una sintaxis basada en camelCase para eventos, como **onClick**, **onChange**, **onSubmit**, etc.

Los manejadores de eventos se pasan como funciones en **JSX**

```
function Button() {  
  const handleClick = () => {  
    alert('¡Botón presionado!');  
  };  
  
  return <button onClick={handleClick}>Presiona aquí</button>;  
}
```



>Xtratego
ACADEMY



Unidad 2.5

Formularios Controlados y No Controlados

Los formularios no controlados utilizan referencias a los elementos del **DOM** directamente para manejar los datos de entrada. En este caso, **React** no gestiona el valor del campo.

```
function Form() {  
  const inputRef = React.useRef();  
  
  const handleSubmit = (event) => {  
    event.preventDefault();  
    alert(`El nombre ingresado es: ${inputRef.current.value}`);  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <label>  
        Nombre:  
        <input type="text" ref={inputRef} />  
      </label>  
      <button type="submit">Enviar</button>  
    </form>  
  );  
}
```

En los formularios controlados, los datos de entrada del formulario están vinculados al estado del componente. Esto permite controlar el valor de los campos de formulario mediante **React**

```
function Form() {  
  const [name, setName] = useState('');  
  
  const handleChange = (event) => {  
    setName(event.target.value);  
  };  
  
  const handleSubmit = (event) => {  
    event.preventDefault();  
    alert(`El nombre ingresado es: ${name}`);  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <label>  
        Nombre:  
        <input type="text" value={name} onChange={handleChange} />  
      </label>  
      <button type="submit">Enviar</button>  
    </form>  
  );  
}
```

Ejemplo: [modulo2/calcularedad-react](#)



>Xtratego
ACADEMY



Unidad 2.6

Ejercicio Práctico

Crear un **componente** que muestre un contador con **dos botones**: uno para **incrementar** y otro para **decrementar** el valor del contador. Añadir un mensaje que se **actualice** dinámicamente en función del valor del **contador**

```
function Counter() {  
  const [count, setCount] = useState(0);  
  return (  
    <div>  
      <p>El valor del contador es: {count}</p>  
      <button onClick={() => setCount(count + 1)}>Incrementar</button>  
      <button onClick={() => setCount(count - 1)}>Decrementar</button>  
    </div>  
  );  
}
```