

Implementar eventos en un componente de clase

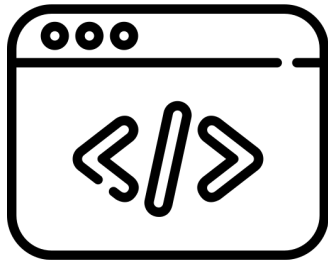
DigitalHouse>



**Certified
Developer**

The Ultimate Tech Degree

Formas de implementar eventos en un componente de clase



React, junto con la continua evolución de JavaScript (ECMAScript), brindan variadas y flexibles formas de desarrollar, actualizar o debuggear una aplicación. En este caso explicaremos tres formas muy extendidas de implementar eventos, todas válidas.

Forma 1

```
class Eventos extends Component {  
  constructor(props) {  
    super(props);  
    this.state = { color: "red" }  
    this.handleChangeColor = this.handleChangeColor.bind(this);  
  }  
  
  handleChangeColor () {  
    this.setState({ color: "blue"});  
  }  
  
  render() {  
    return (  
      <button onClick={this.handleChangeColor}>{this.state.color}</button>  
    );  
  }  
}
```

En este caso tenemos un botón con un evento onClick que llama al manejador handleChangeColor. En esta función se setea la pieza de estado de red a blue.

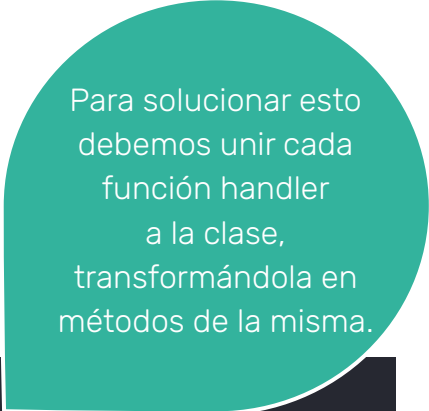
Forma 1 (cont.)

Ahora bien, este armado de por sí no funcionará.

En un componente de clase, el valor de `this` dentro de una función dependerá de cómo fue invocada la función, su contexto. En el caso de ser llamada desde un evento en JSX, `this` apuntará a `undefined` y no a la clase.

Como resultado obtendremos el siguiente error:

Cannot read property 'setState' of undefined

A teal speech bubble with a white border, containing text about binding functions to the class.

Para solucionar esto
debemos unir cada
función handler
a la clase,
transformándola en
métodos de la misma.

```
this.handleChangeColor = this.handleChangeColor.bind(this);
```

Forma 2

```
class Eventos extends Component {  
  state = { color: "red" }  
  
  handleChangeColor = () => {  
    this.setState({ color: "blue" });  
  }  
  
  render() {  
    return (  
      <button onClick={this.handleChangeColor}>{this.state.color}</button>  
    );  
  }  
}
```

¿Qué queremos decir con esto? En una función de flecha, el valor de `this` es determinado por su ámbito que, en este caso, es la clase.

En este caso, nos ahorramos de declarar el constructor y de unir la función con **bind**. Para lograr esto hacemos uso de funciones arrow, las cuales tienen la ventaja de que su scope (ámbito/alcance) es léxico.

Forma 3

En este último caso, la sintaxis declarativa de JSX nos permite —en la misma etiqueta (<button />)— y —desde el evento— llamar directamente a una función arrow anónima que ejecutará las instrucciones.

```
class Eventos extends Component {  
  state = { color: "red" }  
  
  render() {  
    return (  
      <button onClick={() => this.setState({ color: "blue" })}>  
        {this.state.color}  
      </button>  
    );  
  }  
}
```

DigitalHouse>