



Clase 09. React JS

EVENTOS Y DECLARACIONES

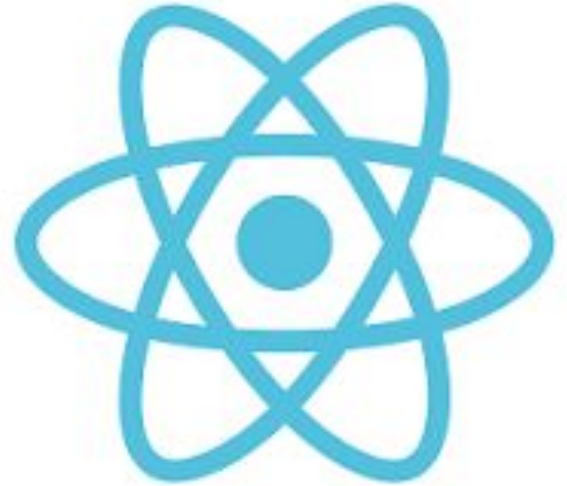


OBJETIVOS DE LA CLASE

- Entender el sistema de eventos de react y su implementación
 - Diseñar componentes orientados a eventos

EVENTOS

Si bien existen muy variados tipos de aplicaciones, es raro encontrar alguna que pueda tener sentido sin **eventos**



¿Qué es un evento en nuestro contexto?

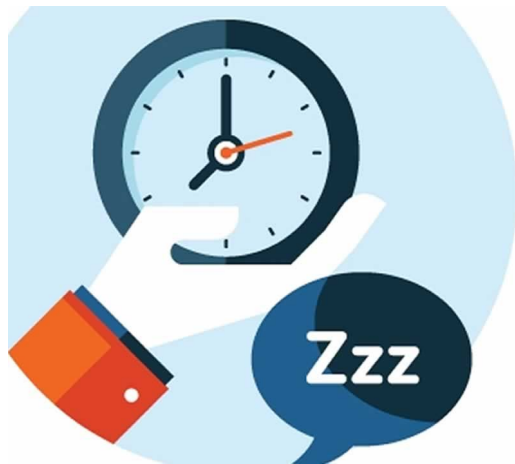
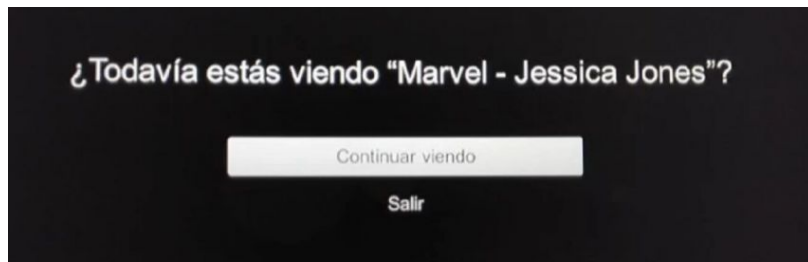
Es un **estímulo** programático que puede ser provocado de manera **automática** o ser el resultado de una **interacción** del usuario con la **UI**



Tipos de eventos

Eventos automáticos


Si estamos viendo **Netflix** por **mucho tiempo** sin tocar el control remoto ocurre un **evento automático por inactividad** que nos pregunta...

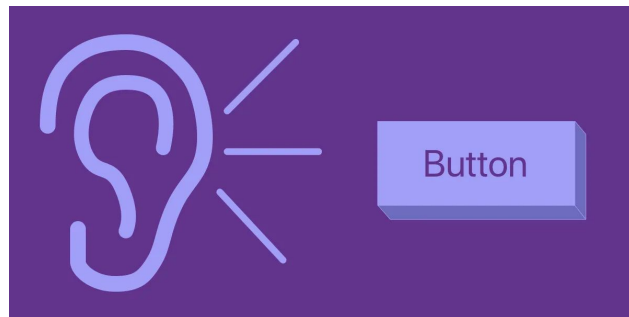


Eventos manuales

Son todas las **interacciones** del usuario que producen algún tipo de **respuesta o efecto** secundario

Text Input

 Please fill out this field.



DOM Events

El **DOM** tiene una serie de eventos estándar y se dividen en varias categorías:

- **Dispositivo/acción:** mouse, input, keyboard, wheel, focus, etc
- **Custom events:** Es posible definir eventos propios que disparen la información que queramos

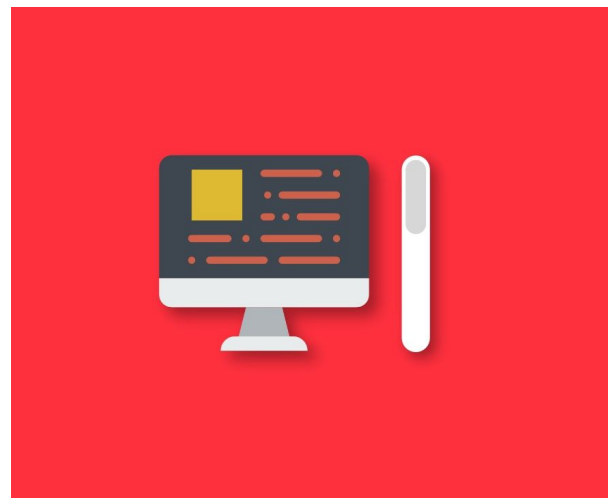


El evento de UI más conocido es el **click**

```
return (  
  <div>e  
    <button onClick={input}>Button</button>  
    <p>Pressed?</p>  
  </div>  
);
```



Aunque probablemente lo utilices casi tanto como el scroll vía un **wheel event**



Event listener

Event listener

Un **Event Listener** es un patrón de diseño que sirve, como su nombre lo indica, para **escuchar cuando un algo ocurre en algún elemento, librería o API**, y poder realizar una acción en consecuencia.



Tip: ¡Hay otros lenguajes que también implementan eventos!

CONFIGURANDO EVENT LISTENERS

Agregando un event listener

```
window.addEventListener('resize', onResize);
```

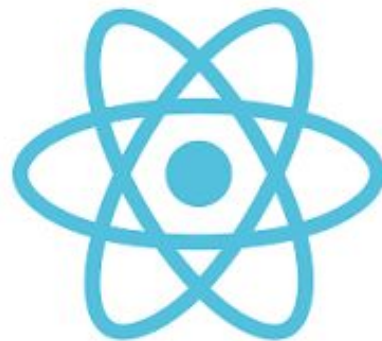
Nombre de evento que quiero escuchar

Referencia de la función a registrar

*Nota: ¡Guardar **referencia** para poder removerlo después!*

Removiendo un event listener

```
return () => {  
  console.log('On dismount');  
  window.removeEventListener(onResize)  
}
```



*Nota: Invocar el **removeEventListener** en la función de limpieza de nuestros hooks en donde los hayamos registrado*



Removiendo eventos

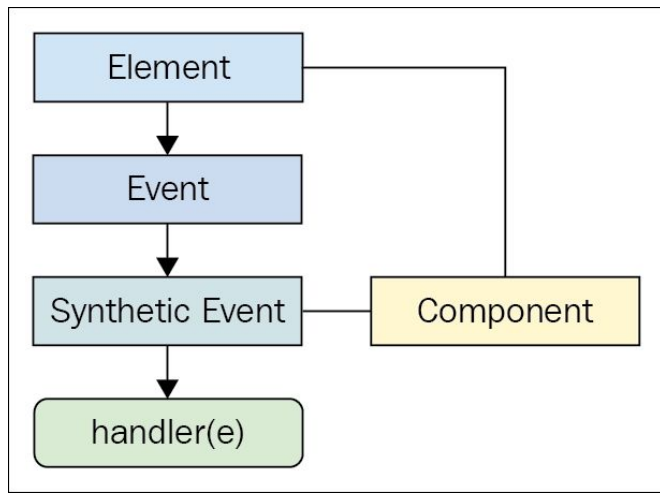
- Se **des-registran** con el nombre y la referencia a la función con que los registramos (no alcanza únicamente el nombre)
- Si registras **manualmente** un evento del **DOM** en tu componente de react hazlo dentro de un **effect** y asegúrate de de-registrarlo en la función de limpieza en el **return** del efecto
- **Recordemos:** Si dejamos event listeners sin des-registrar corremos riesgos de crear leaks de memoria o registrar un evento más de una vez (se ejecutara una vez por cada register)

REACT y los Eventos

Synthetic Events

Los **distintos browsers** suelen tener algunas **variaciones** en el contenido de los eventos.

Esto haría difícil utilizarlos de manera uniforme en cada plataforma. React es consciente de esto y nos ayuda proveyendo esta **abstracción**





Synthetic Events

- Sirven para **normalizar/estandarizar** eventos entre browsers
- Siempre que registre un evento vía React/Jsx con **onClick**, no obtendré el evento nativo, sino uno sintético
- **Se destruyen** al terminar la ejecución de la función vinculada (por performance)
- Puedo acceder al evento nativo via **evt.nativeEvent**

```
> evt.nativeEvent  
◁ MouseEvent {isTrusted: true, screenX: 1062, screenY: 256, clientX: 45, clientY: 19, ...}
```

VAMOS AL CÓDIGO



CODER HOUSE

Declarando un evento

Declarando un evento

```
import React from "react";
import "./style.css";

export default function App() {
  function onClick(evt) {
    console.log('Clicked')
    // Al terminar esta función el evt se destruye
  }
  return (
    <div>
      <button onClick={onClick}>Click-me</button>
    </div>
  );
}
```

Click-me

Console



☒ Clear console on rel

Console was cleared

Clicked



Si necesito almacenar el valor del evento puedo guardarlo en un estado.

Declarando un evento - Precaución

```
import React from "react";
import "./style.css";

export default function App() {
  function onInput(evt) {
    console.log(`Event is: ${evt.target.value}`);
    setTimeout(() => {
      console.log(`Event is destroyed: ${evt.target.value}`);
    }, 100);
    // Al terminar esta función el evt se destruye
  }
  return (
    <div>
      <label>Coder name </label>
      <input onInput={onInput}></input>
    </div>
  );
}
```

Error in ~/src/App.js

(12:61)

Cannot read property 'value' of

Console

☒ Clear console on reload

Console was cleared

Event is: c

Warning: This synthetic event is reused for performance reasons. If you're seeing this, you're accessing the property `target` on a released/nullified synthetic event. This is set to null. If you must keep the original synthetic event around, use event.persist(). See <https://fb.me/react-event-pooling> for more information.


✖ Error: Cannot read property 'value' of null

Si necesito **almacenar** el valor del evento puedo **guardarlo en un estado** o llamar a **event.persist()**, pero no suele ser conveniente y/o necesario.

Cancelando el evento

Algunos eventos como **onKeyDown** son cancelables, por ejemplo:

evt.preventDefault()



```
export default function App() {  
  function onInput(evt) {  
    evt.preventDefault();  
  }  
  return (  
    <div>  
      <label>Coder name </label>  
      <input onKeyDown={onInput}></input>  
    </div>  
  );  
}
```

Evitando la propagación

Los eventos por default se ejecutan en el elemento y en cada uno de sus ancestros. Si esto puede traer algún efecto secundario podemos cancelar la propagación (**bubbling**):

evt.stopPropagation()

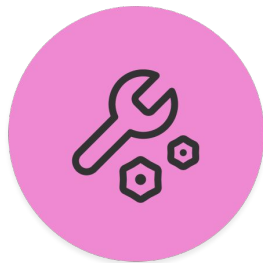


```
export default function App() {  
  function onClick(evt) {  
    evt.stopPropagation();  
  }  
  return (  
    <div>  
      <label>Coder name </label>  
      <input onClick={onClick}></input>  
    </div>  
  );  
}
```

VAMOS AL CÓDIGO



CODER HOUSE



Crea una máscara de input

En [stackblitz](#) crea un input de texto que no permita el ingreso de vocales, cancelando su evento **onKeyDown** en los keys adecuados

***Pista:** el synthetic event de keydown tiene varias propiedades, encuentra cuál te puede dar la información de la tecla ;)*

Tiempo: 15 minutos

CODER HOUSE

¿PREGUNTAS?



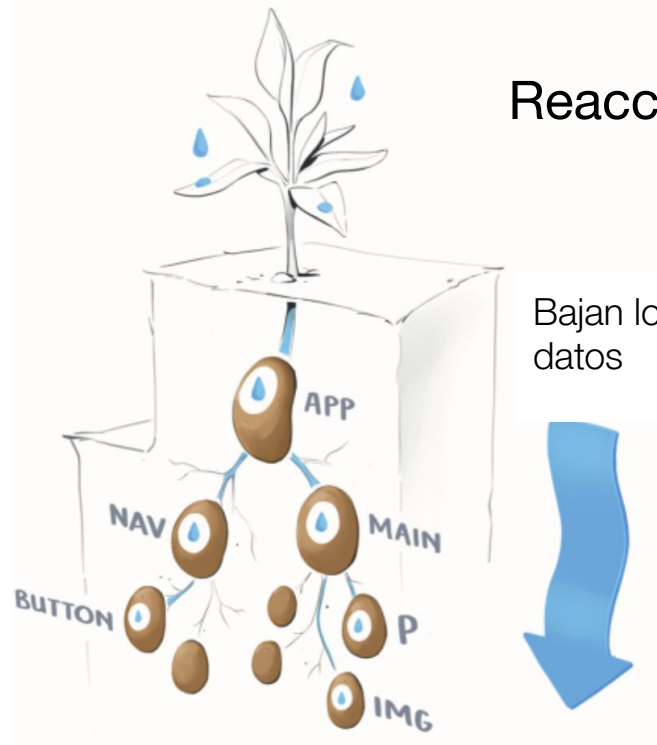
BREAK



CODER HOUSE

COMPONENTES BASADOS EN EVENTOS

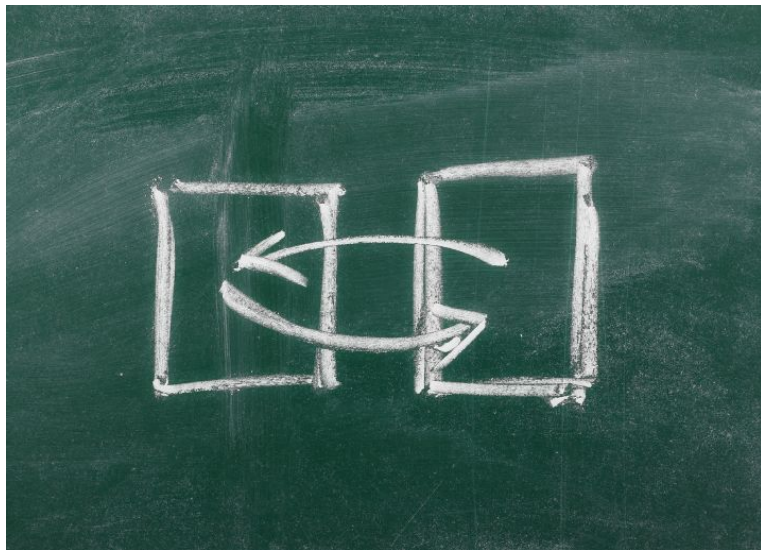
UNIDIRECTIONAL SYMMETRY



***Intercambiabilidad/
agnostic behavior***

Intercambiabilidad

Implementando
componentes de manera
eficiente podremos generar
intercambiabilidad e
intercambiar
funcionalidades sin mucho
esfuerzo



Intercambiabilidad

Podemos generar **variaciones del mismo componente** con distinto layout y el mismo comportamiento

```
const InputCount = (onConfirm, maxQuantity) => {  
  };  
const ButtonCount = (onConfirm, maxQuantity) => {  
  };  
  
export default function ItemDetail({ item, inputType = 'input' }) {  
  const Count = inputType === 'button' ?  
    ButtonCount : InputCount;  
  const itemMax = item.max;  
  const min = item.min;  
  
  function addToCart(quantity) {  
    if(item.inStock) {  
      console.log(`Agregar al cart el item: ${item.id}  
      con cantidad: ${quantity}`);  
    }  
  }  
  
  return (  
    <div>  
      <label>Item description </label>  
      <Count onConfirm={addToCart} maxQuantity={item.max}></Count>  
    </div>  
  );  
}
```

Abstracción - Caso 1

```
function Select({ options, onSelect, defaultOption }) {  
  return <select  
    onChange={(evt) => onSelect(evt.target.value)}  
    {options.map(o => <option value={o.value}>{o.text} </option>)}  
  </select>  
}  
  
export default function App() {  
  const [option, setOption] = useState(1);  
  const options = [{ value: 1, text: 'Azul' }, { value: 2, text: 'Rojo' }];  
  
  function optionSelected(value) {  
    setOption(value)  
  };  
  
  return <>  
    <Select  
      options={options}  
      onSelect={optionSelected}  
      defaultOption={1}>  
    </Select>  
  
    <p>Seleccionada: {option}</p>  
  </>  
}
```

Azul ▾

Seleccionada: 1

Sirve como estrategia
para **ocultar el
comportamiento interno**
de rendering e
implementación de change
events

Console

CODER HOUSE

Abstracción - Caso 2

```
function Select({ options, onSelect, defaultOption }) {  
  return options.map(o => (<  
    <input onChange={(evt) => {  
      onSelect(o.value)  
    }}  
    type="radio"  
    name="color"  
    checked={defaultOption === o.value}  
    id={o.value} />  
    <label for={o.value}>{o.text}</label>  
  </>));  
}
```

```
export default function App() {  
  const [option, setOption] = useState(1);  
  const options = [{ value: 1, text: 'Azul' }, { value: 2, text: 'Rojo' }];  
  
  function optionSelected(value) {  
    setOption(value)  
  };  
  
  return <>  
    <Select  
      options={options}  
      onSelect={optionSelected}  
      defaultOption={option}>  
    </Select>  
  </>
```

☒ Azul ☐ Rojo

Seleccionada: 1

El consumer sigue sin
cambiar la firma de
consumo:

- **onSelect**
- **defaultOption**



Orientación a eventos

- Permite mover la lógica compleja a componentes de menor orden
- Si ambos se comportan igual, el parent no lo sabrá aunque sus implementaciones sean distintas
- Permite que el parent se encargue del resultado final sin darle esa responsabilidad a sus children

VAMOS AL CÓDIGO



CODER HOUSE

¿PREGUNTAS?





Sincronizar counter

Configurar un evento entre el ItemDetail.js y su counter (ItemCount.js del entregable 4) que está dentro de él. Cada vez que se actualice el counter, el botón de compra (Abajo del counter) debe cambiar su texto a **Comprar X**, donde x es la última cifra emitida por el contador.

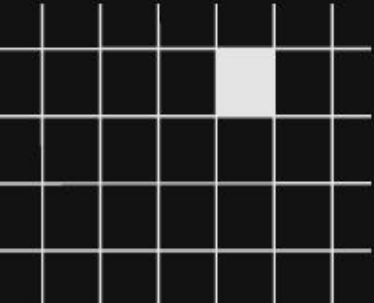
Formato de entrega: GIF mostrando la sincronización parent child en el botón

Tiempo total: 30 minutos



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- DOM & Synthetic Events
 - Diseño de eventos en componentes
- 



OPINA Y VALORA ESTA CLASE