

Pasaron cosas...

"La calidad del software comienza con la calidad de los requerimientos."

Pearl Zhu – Ejecutiva global, autora.

En la vida real

Ocurren a nuestro alrededor, todo el tiempo, pueden ser espontáneos o planificados, creados por nosotros/as, por otras personas, o por factores externos a nuestro control. ¡Estamos hablando de los [eventos](#)! Y este será el tema a abordar en esta toolbox y en la próxima meeting.

En la vida real un evento es la ocurrencia de una o varias acciones las cuales suceden en un momento determinado. ¿Qué es lo primero que se te viene a la cabeza cuando lees la palabra "evento"? Quizás un recital de tu artista favorito, o quizás una conferencia sobre React, un [meetup](#), o una charla sobre frontend en tu comunidad local de desarrollo.



Pero, un evento es más que eso, puede ser la ruptura de un vaso de vidrio que cayó al piso, una llamada telefónica de un familiar, o un encuentro fortuito con una amistad por la calle que tenías tiempo sin ver. Después de todo, un evento es la ocurrencia de una o varias acciones que pasan en un momento. Ah, pero entonces, ¿la vida es una secuencia de eventos? ¡Podría ser!



En la vida virtual

El concepto de **eventos** también está presente en la “vida virtual”, y en esta toolbox te queremos mostrar cuál es su rol dentro de las aplicaciones web, específicamente dentro del frontend, que es el tema que nos compete en este programa.

Así como en la vida real, en una aplicación web, un evento es la ocurrencia de algo que podría pasar. Haz una pausa en la lectura y piensa, ¿qué ejemplos de eventos pueden ocurrir dentro de una aplicación web?

Como te comentamos anteriormente, si en la vida real todo es un evento, en una aplicación web, ¡también lo es! Desplazar el mouse por una pantalla es un evento, hacer click sobre un botón, escribir una tecla en un input, *scrollear* una vista, marcar un checkbox y/o reescalar la ventana de un navegador, son todos ejemplos de **eventos** en una aplicación web.



Manejar eventos con Javascript

Nuestro objetivo es construir aplicaciones web dinámicas que ofrezcan interacciones a las personas usuarias, por lo que el **manejo de eventos** es esencial dentro de nuestra disciplina, el frontend. Los eventos surgen de necesidades de usabilidad y se codean luego en la aplicación, por lo que es un resultado de un trabajo en conjunto entre el equipo de UX/UI y el del frontend.

Manejar un evento no es más que construir la serie de acciones que se desplegarán cuando se de la ocurrencia de este evento. Para manejarlo hay que considerar dos cosas:

1. **Definir el tipo de evento que queremos “escuchar”.** En esta toolbox vamos a manejar eventos sobre elementos HTML, es decir, que requieren la interacción de una persona para que este se ejecute. Existen otros



eventos que no requieren de la acción de una persona, como por ejemplo, eventos que se accionen transcurridos un tiempo determinado, o respuestas fallidas, producto de una petición a un servidor. La definición del tipo de evento consiste en escribir, dentro de la etiqueta HTML en la que queramos agregar un evento, el atributo que indique el nombre del mismo. [Acá puedes encontrar un listado de eventos disponibles](#), organizado en categorías.

2. Una vez definido el evento, deberemos **codear la secuencia de acciones** que se darán, una vez ocurrido el evento. Las acciones las codeamos a través de una función de Javascript.

Cuando hablamos de “escuchar” eventos, nos referimos a la puesta en marcha de los dos puntos anteriores. Es decir, definir el tipo de evento sobre el elemento que actuará como disparador, y la programación de las instrucciones que se ejecutarán una vez accionado el elemento.

En el mundo real, el ejemplo anterior podría ser descrito con la implementación del timbre de una casa. El elemento disparador sería el botón que presionas, y las instrucciones serían los pasos necesarios para que suene, una vez presionado. Decimos que ese botón está siempre “escuchando”, o atento a un evento.

El siguiente video te muestra una forma sencilla de cómo manejar el evento click de un botón. No es necesario utilizar React, ya que la teoría de manejo de eventos forma parte de los controles esenciales de HTML, CSS y Javascript, es decir, es independiente de cualquier librería de front.

[Manejando eventos con Javascript](#)

Los eventos pueden ser manejados para cualquier elemento HTML que exista en la página, siempre y cuando tenga dimensiones que le permita estar visible dentro de la misma. Un evento asociado a un elemento con `display:none` no nos será muy útil.

¡Te proponemos un mini challenge! Partiendo del [sandbox del video](#), crea tu propio botón, **pero** usando la etiqueta `<div>`, en vez de `<button>`, (esto te lo pedimos para que veas que un evento puede ser manejado para cualquier elemento HTML). Para este elemento maneja los siguientes eventos:

- Mostrar un mensaje por consola cuando se haga **dobles click** en el elemento



- Mostrar un mensaje por consola cuando el **mouse entre** al elemento, y también, cuando **salga** de este.



Tip: Para cada evento deberás de crear una función independiente.

Manejar eventos con React

En lo anterior te mostramos una forma sencilla de cómo escuchar por eventos dentro de una aplicación web con Javascript. Como te mencionamos en el video, la teoría de eventos es agnóstica a cualquier librería o framework frontend, y pertenece más bien a lo que Javascript tiene para ofrecer como lenguaje.

Sin embargo, cuando estemos construyendo aplicaciones web con React, como lo estaremos haciendo en la próxima meeting, te contaremos cuáles son las consideraciones que hay que tener en cuenta con JSX y Javascript. **Spoiler alert:** no es muy distinto a manejar eventos con Javascript. De fondo seguiremos utilizando el lenguaje para el manejo de eventos, con unas diferencias sintácticas.

Extra: arrow functions

Hasta ahora te hemos enseñado solo una forma de crear funciones dentro de Javascript, utilizando la palabra reservada `function` de esta manera:

```
...  
  
function manejarElClick(argumentos) {  
    // código de la función  
}  
...
```

Pero, en los últimos años, Javascript propuso una forma distinta de crear funciones, llamada **arrow function**. Este tipo de funciones tienen diferencias muy particulares con la **función declarada** (mostrada en el código anterior), siendo la principal, la sintaxis. Una arrow function se ve de esta manera:

```
...  
  
let manejarElClick = (argumentos) => {  
    // código de la función  
}
```



...

¿Puedes identificar las diferencias sintácticas? El siguiente video explora un poco más las diferencias entre estos dos tipos de funciones.

[Arrow functions](#)

Extra: hoisting

Aparte de las diferencias sintácticas entre las funciones declaradas y las arrow functions, una de las características que separa a estos dos tipos de funciones es que el lugar donde se definen, importa.

En Javascript, una función declarada puede ser definida después de que se invoca, pero una arrow function, no. Por ejemplo, en el siguiente código la función `sumar` es definida después de su invocación:

...

```
sumar(2,2)
// mostraría por consola el número 4
function sumar(x,y) {
  console.log(x + y)
}
```

...

Esto es gracias al concepto de [hoisting](#). En español, la palabra “hoisting” significa “levantar”, y en Javascript, este concepto significa que todas las definiciones de funciones que estén escritas en nuestro código, se van a “levantar” a la línea 0 de nuestra aplicación para que estén disponibles ante cualquier invocación. Esto es una decisión de diseño de Javascript, y últimamente, un feature del lenguaje.

Sin embargo, no es el caso con las arrow functions. Para estas, sí importa el lugar donde estén declaradas, ya que no son “hoistiadas”, o subidas a la línea 0. El siguiente video explora un poco más sobre esta diferencia.

[Hoisting en Javascript](#)



La próxima meeting


En la toolbox de hoy te presentamos una introducción al manejo de eventos con Javascript. Te mostramos cómo escuchar un evento, así como definir las acciones que se realizarán una vez ocurrido el hecho.

Además, te comentamos un par de conceptos de Javascript: las **arrow functions** como una alternativa a escribir funciones declaradas; y el **hoisting**, un feature de Javascript que permite definir funciones declaradas en cualquier parte del código y usarlas antes de su definición.


En la próxima meeting vamos a aprender cómo manejar eventos con React, poniendo en práctica los conceptos aprendidos en esta toolbox. Retomaremos el ejercicio de los álbumes musicales, así como el taskineitor, en su versión 5.0


¡Prepárate para la próxima meeting!


Profundiza

 En la toolbox de hoy te mostramos cómo especificar un evento dentro de una etiqueta HTML a través de un atributo. [En esta guía](#) encontrarás una forma distinta, utilizando solo Javascript, de agregar un evento a un elemento, a través de la función `addEventListener`.

Challenge

 1. Transforma las siguientes funciones declaradas a arrow functions en [este sandbox](#).

 2. [Desde este sandbox](#) agrega un evento para que cuando acaricies al gatito, diga “¡miaul!”. Utiliza arrow functions.

 3. Realiza una aplicación que muestre la cantidad de veces que un cuadrado ha sido presionado. Partiendo desde [este sandbox](#), maneja los eventos `onclick` para cada `div`. Al presionar cada `div`, se deberá mostrar un mensaje por consola que indique la cantidad de veces que todos los `divs` han sido presionados. Utiliza arrow functions. Consulta [este gif](#) para más información.





4. Partiendo de [este sandbox](#), agrega el event handler necesario para manejar el evento cuando el cursor esté sobre el cuadrado.

Al pasar el cursor sobre el cuadrado, se debe mostrar por consola el mensaje **“Disculpe mesero, ¡hay una cruz en mi sopa!”**. Utiliza arrow functions.

