COLOMBIA POTENCIA DE LA VIDA

TIC

Módulo 1

LECCIÓN 4

Eventos, escuchadores y peticiones

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

Universidad Tecnológica de Pereira

Eventos

Los eventos son cosas que suceden en el sistema que se está programando. El sistema se encarga de producir una señal de cierto tipo cuando un evento ocurre, y proporciona un mecanismo para que una acción se lleve a cabo (ejecutar código) de forma automática cuando el evento ocurra. Los eventos se lanzan dentro de la ventana del navegador y usualmente están asociados a un elemento en específico dentro de dicha ventana. Esto puede ser un solo elemento, un grupo de elementos, el documento HTML cargado la pestaña actual, o la ventana del navegador en su totalidad. Existen distintos tipos de eventos que pueden ocurrir.

Por ejemplo:

- El usuario selecciona, hace clic o pasa el ratón por encima de cierto elemento.
- El usuario presiona una tecla del teclado.
- El usuario redimensiona o cierra la ventana del navegador.
- Una página web terminó de cargarse.
- Un formulario fue enviado.
- Un vídeo se reproduce, se pausa o termina.
- Ocurrió un error.

Un listado de muchos eventos separados por categorías puede ser encontrado aquí HTML Event Attributes

Unset

```
<button
onClick="alert('Hello!')">Saludar</but
ton>
```

Para gestionar los eventos o asignar un evento a un elemento del DOM se suele utilizar un manejador de eventos. Esto es un bloque de código (normalmente una función de JavaScript que tú como programador codificas) que se ejecuta cuando el evento ocurre. Cuando uno de estos bloques de código se configura para ejecutarse en respuesta de un evento, decimos que estamos **registrando un manejador de eventos.**

La forma más recomendable para añadir un escuchador es hacer uso del método .addEventListener(), el cuál es mucho más potente y versátil para la mayoría de los casos.

JavaScript

```
const button =
document.querySelector("button");
button.addEventListener("click",
function() {
 alert("Hello!");
});
```

Delegación de eventos

En estos casos se está aplicando un escuchador de un evento click del mouse directamente al botón, pero puede existir el caso en el que queramos saber si el usuario realizó una acción sobre cualquier parte de la página y con base en dicha información realizar alguna acción.

Un ejemplo sencillo es saber si al momento de hacer click sobre el botón, el usuario mantiene presionada la tecla CTRL, para simular un click derecho

JavaScript

```
const button =
document.querySelector("button");
const eventManager = {
 handleEvent: function(ev) {
  if (ev.type === "click") {
   alert("¡Has hecho click!");
```

```
  } else if (ev.type === "mouseleave") {
   alert("¡Has abandonado el botón!");
  }
 }
button.addEventListener("click",
eventManager);
button.addEventListener("mouseleave",
eventManager);
```

El objeto/variable “ev” de la función almacena toda la información del evento que acaba de dispararse, por ejemplo accediendo a “ev.target” es posible obtener información sobre qué elemento del DOM se aplicó la acción, en este caso el click.

Una forma de conocer todas las posibilidades con los eventos es imprimir el objeto “ev” en consola y consultar sus propiedades. **evento - Referencia de la API Web | MDN**

FETCH

Para trabajar con fetch y hacer algunas ejercitaciones, es necesario utilizar algunas herramientas externas, una de las más sencillas de utilizar es Giphy, para ello es necesario crear una cuenta en la sección de desarrolladores **https://developers.giphy.com/dashboard/**
Siempre es importante contar con la documentación oficial de las herramientas para facilitar su aprendizaje, en el siguiente enlace puede encontrar una excelente guía.

La idea aquí es que el docente de la mano con los estudiantes hagan el procedimiento de creación de la cuenta y su primer proyecto para que obtengan el código o llave de la API, este es un muy buen ejercicio para hacer peticiones de tipo GET y hacer una página web que muestre GIFs.

Para probar el método post, la siguiente página es una excelente alternativa, aunque también cuenta con métodos GET, son solo de texto, **JSONPlaceholder.**

FETCH es una función que permite hacer peticiones asíncronas, como comunicarse con APIs o consultar datos de un JSON. Se pueden enviar pedidos mediante el método ****GET**** o mediante el método ****POST****.

GET

En caso que el pedido sea por ****GET****, ****fetch()**** recibe un solo parámetro, que será el endpoint (URL).
La función es asíncrona (no sabemos cuándo va a responder el servidor), por lo tanto **fetch()** devuelve una promesa, por lo tanto utilizamos **then()**
Siempre va a recibir dos .then(). Se van a utilizar promesas anidadas.

En el primer .then() se va a resolver el pedido al servidor. Como generalmente las APIs retornan la información en formato JSON, por eso una vez que respondan, lo más común es decodificarlas con el método **.json()** devuelve la promesa a decodificar en la respuesta.

Unset

```
fetch(url)

    .then(response => response.json)
```

Como el primer .then() devuelve una promesa, necesitamos usar otro debajo para trabajar en él, con la info ya decodificada.

Unset

```
fetch(url)

        .then(response => response.json)
```

```
        .then(dataDecoded =>
console.log(dataDecoded));
```

Si por ejemplo el servidor con el que se comunica en el endpoint dejó de darle soporte a la API,se recibiría un error, por lo cual podemos también utilizar el método **.catch()** como en todo pedido asíncrono.

Unset

```
fetch(url)

        .then(response => response.json)

        .then(dataDecoded =>
console.log(dataDecoded))

        .catch(err => console.log(err))
```

Ejemplo API de Gifs

Unset

```
window.addEventListener('load', ()=>{

    fetch(urlApiGiphy)

        .then(response => response.json())

        .then(info =>{
```

```
                console.log(info.data)


                info.data.forEach(data =>{

                    let gif = '<p>' + data.title +
'</p>'

                    gif += '<img src=
'data.images.orinial.url + '>'

document.querySelector('ul').innerHTML +=
'<li/>' + gif + '</li>'

                })

                .catch(e => alert('error!!')

};
```

POST

Cuando lo utilizamos con POST, y como enviaremos datos, debemos pasarle dos parámetros, el primero sigue siendo la URL de la API, y en el segundo se configura un objeto literal con los datos necesarios para que la API entienda la petición.

```
Unset
fetch(url, {

        method : 'POST',

        body : JSON.stringify(data) //esta
variable originalmente era un objeto literal

        headers : {

                'Content-Type' :
'application-json'

                }

})
```

Lo primero a definir es el método que se utilizará, en este caso el método **POST**.

El segundo atributo es el más importante, se va a llamar body y tendrá el contenido del envío que siempre deberá estar en formato JSON, y es por eso que la enviamos con JSON.stringify().

El último atributo a configurar es el headers o cabeceras, que recibe el tipo de contenido que vamos a enviar para que pueda ser interpretado por el servidor que recibe la respuesta.

Ejemplo API de Gatitos

```
Unset
window.addEventListener('load', ()=>{

        //lo primero es armar la información que
queremos ENVIAR a la API
```

```
        //este esquema va a depender de lo que pide
la API.

        let data = {

                image_id : "asf2",

                sub_id : "user-123",

                value : 1

        };


        //acá armamos una variable de
configuración

        let settings = {

                'method' : 'POST',

                'body' : JSON.stringify(data),


                //encabezados que necesita la api, a
veces son optativos, depende de la API

                'headers' : {

                        'Content-Type' :
'application-json',

                        'x-api-key' : '.....apiKEY'

                }

};

        fetch(urlApiGatitos, settings)
```

```
        .then(response => response.json())

        .then(info => console.log(info))

                .catch(e => alert('error!!')

};
```

Ejercitación

Vamos a usar FETCH por GET

Es hora de poner en práctica lo visto hasta ahora. Si bien parece que usar fetch es sencillo, se debe hacer al menos una vez. ****Para eso se tiene que:

- Escribir la función de un fetch que ejecute un GET en la URL: 'https://api.chucknorris.io/jokes/random'.
- En el primer then, retornar el JSON de lo que se recibió por parámetro.
- En el segundo then, hay que imprimir por consola lo que recibe esa función por parámetro. ¡A la práctica!

Unset

```
fetch('<https://api.chucknorris.io/jokes/rand
om>')

  .then(response=>{

    response.json();

  })

  .then(info=>{

    console.log(info)

  })
```