educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 19

Operaciones

Petición GET con AJAX

Es una petición HTTP que busca obtener datos de otra URL. **Estas peticiones son de solo lectura,** y pueden solicitar tanto recursos externos como internos al sitio.

Casos de uso

- Obtener un JSON de una API REST externa.
- Carga retardada *(lazy-loading)* de páginas HTML.
- Carga retardada *(lazy-loading)* de recursos de hipermedia (imágenes, audios, videos).

Ejemplo 1

```
function getJson({url, onLoad})
{
    const xhr = new XMLHttpRequest();
    /**
     * AL configurar el responseType como 'json'
     * se ejecutará la siguiente expresión obtener
     * la respuesta:
     *
     * xhr.response = JSON.parse(xhr.response)
     */
    xhr.responseType = 'json';
    xhr.open('GET', url);
    xhr.send();
    xhr.addEventListener('load', () => render(xhr.response));
}

getJson({
    url: 'https://jsonplaceholder.typicode.com/users',
    onLoad: (content) => {
        console.log(content);
    }
});

```

Ejemplo 2

```
function getDocument({url, onLoad})
{
    const xhr = new XMLHttpRequest();
    /**
     * Al configurar el responseType como document
     * se leera el contenido del archivo y se generará un
     * DOM con su contenido. El archivo debe tener un solo nodo raíz
     */
    xhr.responseType = 'document';
    xhr.open('GET', url);
    xhr.send();
    xhr.addEventListener('load', () => onLoad(xhr.response));
}

getDocument({
    url: 'https://jsonplaceholder.typicode.com/users',
    onLoad: (data) => {
        console.log(data);
    }
});

```

Ejemplo 3

```
1   function getPage({url, render})
2   {
3       const xhr = new XMLHttpRequest();
4       xhr.open('GET', url);
5       xhr.send();
6       xhr.addEventListener('load', () => render(xhr.response));
7   }
8
9   getPage({
10      url: 'https://jsonplaceholder.typicode.com/users',
11      render: (content) => {
12          document.getElementById('app').innerHTML = content
13      }
14  });
15
```

Petición POST

Es una petición de escritura de datos. Sirve para enviar datos al servidor o a una API, en lugar de recibir datos desde ella. Es importante resaltar que, si bien se realiza una petición de escritura, es eso: una petición.

Se solicita al servidor que haga una acción de escritura de datos, pero **es la lógica del servidor la que determina si dicha petición será aceptada** y qué cambios se harán en las estructuras de datos.

Casos de uso

- Envío de formularios sin recargar la página.
- Login / Signin sin recargar la página.
- Envío de estadísticas y analíticas invisibles a la interfaz de usuario.

Ejemplo 1

```
<form id="Encuesta">
    Usuario:
    <input type="text" name="username" />
    <br />
    Contraseña:
    <input type="password" name="password" />
    <br />
    <button>Iniciar Sesión</button>
    <div id="msj"></div>
</form>
```

Ejemplo 2

```
12  document.getElementById('Encuesta').addEventListener('submit', (evt) =>
13  {
14      evt.preventDefault();
15
16      const xhr = new XMLHttpRequest();
17      xhr.open('POST', '/api/login');
18      /**
19       * Además de cambiar el método en .open()
20       * debemos añadir la información del envío como
21       * argumento a .send(). Este argumento tiene como
22       * tipo de dato XMLHttpRequestBodyInit:
23       * https://fetch.spec.whatwg.org/#typedefdef-xmlhttprequestbodyinit
24       * Ese tipo es un tipo genérico que envuelve texto, blob, formData, etc
25       */
26      xhr.send(new FormData(evt.target));
27      xhr.addEventListener('load', () =>
28          document.getElementById('msj').innerHTML = xhr.response
29      );
30  });
```

Progreso en AJAX

`XMLHttpRequest` proporciona la capacidad de escuchar varios eventos que pueden ocurrir mientras se procesa la solicitud. Esto incluye notificaciones de progreso periódicas, notificaciones de errores, y otros.

El soporte para la supervisión de evento de progreso DOM de las transferencias **`XMLHttpRequest`** sigue la especificación para eventos de progreso: estos eventos implementan la interfaz **`ProgressEvent`**.

La interfaz **`ProgressEvent`** representa eventos que miden el progreso de un proceso subyacente, como una solicitud HTTP (para un **`XMLHttpRequest`**, o la carga del recurso subyacente de un **`<img>`**, **`<audio>`**, **`<video>`**, **`<style>`** o **`<link>`**).

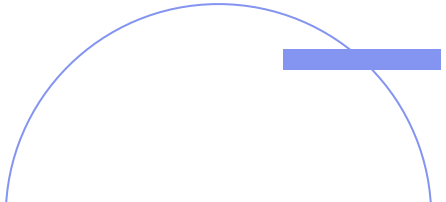
Los eventos de este tipo cuentan con las siguientes propiedades, entre otras:

- **`lengthComputable`:** indicador booleano que muestra si es calculable el trabajo total que se debe realizar y la cantidad de trabajo ya realizado por el proceso subyacente.

  En otras palabras, dice si el progreso es mensurable o no.

- **`loaded`:** es un número sin signo que representa la cantidad de trabajo ya realizado por el proceso subyacente. La proporción de trabajo realizado se puede calcular con la propiedad y **`ProgressEvent.total`**. Al descargar un recurso usando HTTP, esto solo representa la parte del contenido en sí, no los encabezados y otros gastos generales.

- **`total`:** es un número sin signo que representa la cantidad total de trabajo del proceso subyacente. Al descargar un recurso usando HTTP, esto solo representa su contenido, no los encabezados y otros gastos generales.



```
//...
xhr.addEventListener("progress",function(e){
   if(e.lengthComputable){
       console.log(e.loaded,e.total)
   }
})
//...
```

¡Sigamos trabajando!