Api Fetch

AJAX

En muchas ocasiones nuestras aplicaciones deben comunicarse a otro lugar para traer información relevante o de significado para el usuario, generalmente esta petición se realiza a un servidor, para poder establecer esta comunicación es necesario contar con dos elementos para dicha acción.

El primer elemento es el protocolo HTTP (HyperText Transfer Protocol o Protocolo de Transferencia de Hipertexto) el cual permite la comunicación entre sistemas físicamente dispersos. Fue originalmente ideado por Sir Tim Berners Lee en 1989. Ahora está coordinado por el W3C. En su formato más simple, establece el medio de comunicación de las páginas web, partiendo del servidor web al navegador del usuario.

El segundo elemento es AJAX acrónimo de JavaScript Asíncrono + XML (AJAX), siendo un conjunto de técnicas de desarrollo web que permiten que las aplicaciones funcionen de forma asíncrona, procesando cualquier solicitud al servidor en segundo plano.

- Son peticiones al servidor que esperan una respuesta. Como un walkie talkie.
- Se solicita información y se espera una respuesta. Por ejemplo Un formulario de login.
- Se usa para consumir APIs y recursos web.
- Ajax usa el protocolo HTTP.
- Es una conexión de una sola vía.

Definición de Api Fetch

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas.

El método global fetch() proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red. Anteriormente era usado XMLHttpRequest.

El elemento fetch recibe como parámetros dos elementos, el primero es el url del recurso y el segundo un objeto literal llamado options.

El parametro options tiene varias propiedades, sin embargo no entonos los escenarios se utilizan, sus principales elementos son:

Propiedad	Definición
method	Define el método http a utilizar, en el caso de Get se puede omitir.
headers	Establece las cabeceras de la petición
body	Presente en en los métodos HTTP Post, Put y Patch, es la información enviada del cliente al servidor.

Ejemplo de uso de fetch

```
fetch(url,{
    method: 'POST', // *GET, POST, PUT, DELETE
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(data) //Presente en métodos POST Y PUT
});
```

La interfaz fetch por definición es una promesa, esto nos dice que tendremos los métodos then en caso de que la respuesta se satisfactoria y catch cuando se suscite un error.

Como ejemplo ilustrativo realizaremos una petición get.

```
fetch('http://example.com/movies.json')
 .then(response => response.json())
 .then(data => console.log(data))
 .catch(error=>console.log(error));
```

Métodos (Verbos) HTTP

Los métodos o verbos http son un conjunto de elementos que nos permiten declarar las acciones que se desean realizar en un determinado recurso.

Entender este concepto es fundamental para comprender la forma en que funciona la arquitectura REST, pues mediante los métodos le indicamos al servidor la forma en que debe de tratar una determinada petición.

El protocolo HTTP define una gran cantidad de métodos que so utilizados en diferentes escenarios, los más utilizados hasta ahora son los siguientes:

Método	Definición
get	Usado para consultar un recurso, esta petición no causa efectos secundarios en el servidor.
post	Es utilizado para crear un nuevo elemento, Cada llamada con POST debería producir un nuevo recurso.
put	Es usado para modificar un elemento existente.
delete	Usado para eliminar registros, bien pudiera ser para eliminar un recurso individual o varios.

Peticiones Get

Para realizar una petición get en la mayoria de cosas es necesario solo conocer el url, para este ejemplo utilizaremos el api de Rick and Morty para la petición siendo

la siguiente forma.

```
fetch('https://rickandmortyapi.com/api/character')
 .then(response => response.json())
 .then(data => console.log(data))
 .catch(error=>console.log(error));
```

Peticiones Post / Put

En las peticiones Post y Put son muy similares pues en ambas se envía información al servidor, siendo la primera para crear un elemento y la segunda para actualizar el elemento.

Para hacer cualquiera de estas peticiones es necesario definir el objeto literal option en la petición.

Ejemplo de petición POST

```
    fetch('http://example.com/create',{
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    })
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error=>console.log(error));
```

Ejemplo de petición PUT

```
    fetch('http://example.com/update',{
        method: 'PUT',
```

```
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    })
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error=>console.log(error));
```

Peticiones Delete

Este método es utilizado para eliminar un registro existente, al igual que POST y PUT es necesario declararlo en el objeto option del objeto fetch, sin embargo a comparación de los métodos mencionados anteriormente, delete no dispone de un body.

Ejemplo de petición DELETE

```
    fetch('http://example.com/delete',{ method: 'DELETE'})
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error=>console.log(error));
```