

BACK-END I

Node y Express

Tema 5 | Consumo de APIs y Peticiones HTTP

Índice

1. Consumo de APIs Externas
2. Clientes Http
3. Tipos de Clientes
4. Retos

DEFINICIÓN

Consumo de APIs externas

- Como vimos en las lecciones anteriores, podemos definir una api como una **interfaz de comunicación con una aplicación**. De este modo, cualquier aplicación que permita la extracción de los datos de la misma, pondrá a disposición de los usuarios una API para que esta sea consumida.
- Se podría decir, que todas las aplicaciones tienen una API, aunque dependiendo del tipo de aplicación, y sobre todo, dependiendo de los tipos de datos que maneje, **puede ocurrir que estas APIs no sean públicas**.
- Por ejemplo, la mayoría de videojuegos tienen APIs públicas en las que ofrecen los datos del videojuego, pero la información relativa a los usuarios suele ser privada. Por otro lado, los gobiernos también tienen muchísima información disponible en APIs, pero la información sensible, no se publica.

DEFINICIÓN

Consumo de APIs externas

- De este modo, nos encontramos en un panorama en el que simplemente tenemos que **consumir una API de un tercero para obtener sus datos** y poder hacer aplicaciones utilizando esa información
- La principal diferencia entre una API de un tercero y una API propia, es que nosotros (Al menos por ahora) solo hemos trabajado con nuestra **API en local**. Esto significa que nuestra API se estaba ejecutando en un puerto de nuestro ordenador, y nadie puede acceder a ella
- Para publicar nuestra API, lo único que tenemos que hacer es pasarla de local a remoto, es decir, **publicarla en un servidor remoto y obtener un enlace**. Este enlace (URL) es el enlace que utilizarían el resto de usuarios para hacer peticiones a nuestra API

DEFINICIÓN

Consumo de APIs externas

La forma de consumir una API es relativamente sencilla:

- **Buscar en internet la API** a la que queremos conectarnos. Necesitamos un **enlace**, es decir, la url a la que lanzar las peticiones.
<https://pokeapi.co/api/v2/pokemon/ditto>
https://datos.gob.es/apidata/catalog/dataset?_sort=title&_pageSize=10&_page=0
- **Leer la documentación de la API**, donde podremos ver los diferentes endpoints que tiene y si requiere algún valor para funcionar:
<https://pokeapi.co/docs/v2>
[https://datos.gob.es/apidata - !/dataset/findAllDatasets](https://datos.gob.es/apidata-!/dataset/findAllDatasets)
- **Lanzar una petición**. La petición la lanzaremos o desde la propia documentación de la API o desde Postman para hacer pruebas. Más adelante, la haremos desde nuestra aplicación, ya sea desde el front o el back.
- **Analizar la respuesta** que devuelve la API

| Clientes HTTP

DEFINICIÓN

Cientes HTTP

Como ya mencionamos anteriormente, un cliente es un **programa que vamos a utilizar para enviar peticiones HTTP** a un servidor. Hasta ahora, hemos visto dos clientes, los navegadores en general y Postman.

Estos clientes, sirven para enviar peticiones y ver qué nos devuelven, pero si queremos hacer una app automatizada que haga peticiones cuando el usuario lo necesite, vamos a necesitar otros **clientes que sean más versátiles**.

En JavaScript nativo, existen varios clientes, por lo que vamos a ver cómo utilizarlos y desde dónde se deben lanzar.

CLIENTES HTTP

XMLHttpRequest

El primer cliente de JavaScript que vamos a ver es **XMLHttpRequest**. Es un cliente que se lanza **desde el front** de una aplicación, es decir, desde el controlador.

Este cliente de JavaScript ya no se usa tanto, ya que se ha sustituido principalmente por **Fetch**, que lo veremos más adelante

Dicho esto, es interesante ver cómo funciona y qué operaciones hay que hacer para poder enviar la petición:

- Crear una clase que se adapte al dato si es necesaria.
- Crear un objeto de tipo XMLHttpRequest
- Llamar al método open y pasarle como argumento el tipo de petición y la url de la api
- Enviar los headers
- Enviar la petición

CLIENTES HTTP

XMLHttpRequest

Creamos primero una clase que se adapte al dato:

```
class User
{
    constructor(nombre, apellidos)
    {
        this.nombre = nombre;
        this.apellidos = apellidos
    }
}
```

CLIENTES HTTP

XMLHttpRequest

Veamos cómo funciona una petición de tipo POST:

```
function postUser()
{
    let user = new User(document.getElementById("nombre").value,
                        document.getElementById("apellidos").value)

    console.log(JSON.stringify(user));

    let xhttp = new XMLHttpRequest();

    xhttp.open("POST", "http://localhost:3000/usuario", true);
    xhttp.setRequestHeader("Content-type", "application/json")
    xhttp.send(JSON.stringify(user))
}
```

CLIENTES HTTP

XMLHttpRequest

Petición de tipo GET. Lanzamos la petición y vemos el resultado con onreadystatechange:

```
function getUser()
{
    let xhttp = new XMLHttpRequest();

    xhttp.onreadystatechange = function (aEvt)
    {
        if (xhttp.readyState == 4 && xhttp.status == 200)
        {
            console.log(xhttp.responseText);
            let dataJson = JSON.parse(xhttp.responseText);
            document.getElementById("mostrarNombre").value = dataJson.data.nombre;
            document.getElementById("mostrarApellidos").value = dataJson.data.apellidos;
        }
    }

    xhttp.open("GET", "http://localhost:3000/usuario", true)
    xhttp.send()
}
```

CLIENTES HTTP

Fetch

El cliente Fetch, también sirve para hacer peticiones HTTP **desde el front** y en general, es el más utilizado. La forma de lanzarlo es bastante similar al XMLHttpRequest, con la particularidad de que **su resultado devuelve una promesa**, por lo que es bastante sencillo de analizar. Veamos los pasos necesarios para lanzarlo:

- Crear una clase que se adapte al dato si es necesaria.
- Definir la url de la API
- Definir los headers de la petición y el método
- Lanzar el método fetch utilizando la url y los headers como parámetro
- Analizar su resultado con las herramientas que vimos para tratar promesas:
 - then/catch
 - async/await

Fetch: Creación de la clase

Creamos primero una clase que se adapte al dato:

```
class User
{
  constructor(nombre, apellidos)
  {
    this.nombre = nombre;
    this.apellidos = apellidos
  }
}
```

CLIENTES HTTP

Fetch: Petición POST

```
async function postUser()
{
  try
  {
    let user = new User(document.getElementById("nombre").value,
                        document.getElementById("apellidos").value)

    let url = "http://localhost:3000/usuario";

    let param =
      {
        headers: {"Content-type": "application/json; charset= UTF-8"},
        body: JSON.stringify(user),
        method: "POST"
      }

    let data    = await fetch(url, param);
    let result  = await data.json();

    console.log(result)
  }
  catch(error)
  {
    console.log(error)
  }
}
```

CLIENTES HTTP

Fetch: Petición GET

```
async function getUser()
{
  let url = "http://localhost:3000/usuario";

  let param =
  {
    headers: {"Content-type": "application/json; charset= UTF-8"},
    method: "GET"
  }

  try
  {
    let data = await fetch(url, param);
    let result = await data.json();
    document.getElementById("mostrarNombre").value = result.data.nombre;
    document.getElementById("mostrarApellidos").value = result.data.apellidos;
  }
  catch(error)
  {
    console.log(error)
  }
}
```

CLIENTES HTTP

Axios

Vamos a ver un último cliente para lanzar peticiones HTTP que se llama Axios. Este cliente, se diferencia de los otros dos en que **está pensado para lanzarse desde el back** de una aplicación. En muchas ocasiones, puede ocurrir que no queramos lanzar una petición desde nuestro front sino desde nuestro back (desde nuestra API propia) y para estos casos, utilizaremos axios. Veamos cómo funciona:

- Tendremos que llamarlo en algún sitio de nuestro back, en nuestro caso, lo mejor es **crear un endpoint en nuestra API** y llamarlo desde ahí
- Instalar el módulo axios: `npm install axios`
- Tenemos que **importar el módulo** Axios de node
- Luego, definiremos la URL de la API que queremos consumir
- Por último, llamaremos a axios y al método que queramos utilizar (GET, POST...)

Axios : Ejemplo de Uso

```
app.get("/user",
  function(request, response)
  {
    const url = "http://localhost:3000/usuario";

    axios.get(url)
      .then(function (data)
      {
        response.send(data.data)
      })
      .catch(function (error)
      {
        response.send(error)
      })
  }
);
```

CLIENTES HTTP

Conclusión

Para terminar con este tema, veamos un serie de conclusiones:

- Para consumir una API externa, lo único que necesitamos **es la URL de la API y un cliente** para lanzar peticiones HTTP
- Como clientes de prueba, siempre podemos utilizar el navegador o Postman. Además, en la propia documentación de la API suele haber un cliente para probarla
- Cuando consumamos APIs en nuestras aplicaciones, usaremos los distintos clientes que pone a nuestra disposición JavaScript (**XMLHttpRequest, Fetch, Axios**)
- Es importante **leer bien la documentación de cada API**, ya que en muchos casos requerirá que pasemos algún parámetro en nuestra petición para que funcione
- Las APIs más comunes no requieren autenticación, pero habrá otras que **requieran un token** para poder utilizarlas

Glosario

CONSUMO DE APIS EXTERNAS

XMLHTTPREQUEST

FETCH

AXIOS

#Retos

RETOS

Reto

En este reto, vamos a consumir una API externa:

1. Crea un **front** que conste de un archivo **HTML**, un **JS** y un **CSS**.
En este caso, la maquetación es libre, pero como mínimo debería tener un formulario con un **input** (nombre) y al menos un **botón**
2. Lee detenidamente la documentación de la siguiente API: <https://pokeapi.co/>
3. Crea las siguientes funcionalidades:
 - Crea una función que recoja el nombre de un **Pokemon** y lance una petición a la API de Pokemon. La respuesta se debe mostrar en la pantalla
 - Como mínimo, debe mostrar el **nombre**, **imagen** y las distintas habilidades que tiene en formato **tabla**

OPCIONAL: Investiga el resto de endpoints de la API para añadir funcionalidad a la página

codenotch