

M1 - 10 - JS APIs REST

Sus siglas, provienen de su nombre en inglés Application Programming Interface, y es una puerta de acceso a un servicio de un tercero. Esto nos permitirá acceder y obtener datos externos que podemos utilizar e incorporar a nuestra propia aplicación o página web.

¿Cómo accedemos a los datos de una API?

Lo primero que tenemos que tener en cuenta es que existen muchísimas APIs disponibles, pero cada una funciona de una manera diferente con lo que primero deberemos acceder a la página de la API que queremos utilizar y **leer detenidamente la documentación** dónde nos explica los diferentes servicios que ofrece, las URL asociadas y los datos que podrás encontrar.

Veamos un ejemplo práctico, vamos a crear una página web que nos muestra de manera aleatoria chistes de chuck norris.

Pero ¿Vamos a tener que inventarnos esos chistes? NO, accederemos a una API que nos va a dar esa información, concretamente una que nos devuelve específicamente chistes aleatorios de chuck norris.

Como hemos comentado anteriormente lo PRIMERO que debemos hacer es entrar en la página de la API dónde nos explicará cómo podemos acceder a esa información para incorporarla a nuestra página web.

Fetching a random joke

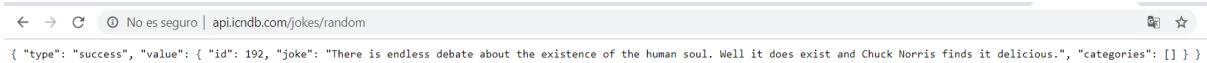
URL:

```
http://api.icndb.com/jokes/random
```

Result:

```
{ "type": "success", "value": { "id": , "joke":  
} }
```

En esta página encontramos un apartado ‘Fetching a random joke’ o lo que es lo mismo ‘Extraer un chiste al azar’, que es lo que queremos hacer nosotros, y vemos que nos proporciona una URL. Las APIs web son las APIs accesibles a través de la Web, utilizando el protocolo REST mediante HTTP o HTTPS, y por ello para hacer peticiones utilizaremos una URL. De hecho, si copiamos esta dirección en el navegador podremos ver lo que nos devuelve.



Podremos utilizar diferentes URL para hacer diferentes peticiones y obtener diferente información.

Por ejemplo podemos ver cómo en el siguiente apartado ‘Fetching multiple random jokes’ - ‘Extraer varios chistes al azar’ nos indica otra URL a partir de la cual podemos obtener no uno sino varios chistes aleatorios a la vez, concretamente los que indiquemos en la URL, como en el ejemplo que estaríamos extrayendo 3 chistes.

Fetching multiple random jokes

URL:

```
http://api.icndb.com/jokes/random/
```

Example:

```
http://api.icndb.com/jokes/random/3
```

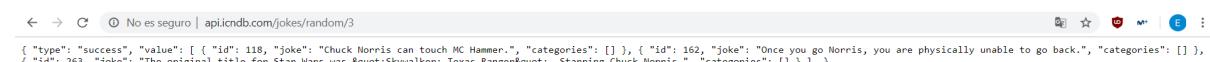
Result:

```
{ "type": "success", "value": &gt; }
```

Example:

```
{ "type": "success", "value": [ { "id": 1, "joke": "Joke 1" }, { "id": 5, "joke": "Joke 5" },
```

Si copiamos la URL en el navegador veremos que efectivamente nos devuelve 3 chistes.



Una forma de observar más fácilmente el resultado de las peticiones es a través de una plataforma que utilizaremos a lo largo del curso que se llama Postman.

Postman

Postman es una herramienta que nos va permitir explorar, probar la API que queremos utilizar, lo que junto a la documentación que encontramos en su página web nos ayudará a saber **qué información nos proporciona la API y en qué formato nos llega estructurada**.

The screenshot shows the Postman application window. In the top bar, there are tabs for 'New', 'Import', 'Runner', and 'APIs'. The main area is titled 'Untitled Request' with a 'GET' method selected. The URL is set to 'http://api.icndb.com/jokes/random'. On the left, there's a sidebar with a 'History' section containing a single entry: 'GET http://api.icndb.com/jokes/random'. Below the request details, there are tabs for 'Body', 'Cookies (1)', 'Headers (15)', and 'Test Results'. The 'Body' tab is active, showing a JSON response with the following structure:

```

1 {
2   "type": "success",
3   "value": {
4     "id": 172,
5     "joke": "Some kids play Kick the can. Chuck Norris played Kick the keg.",
6     "categories": []
7   }
8 }

```

At the bottom right of the response pane, it says 'Status: 200 OK Time: 112ms Size: 747 B Save Response'.

Cuando abrimos el postman, vemos cómo tenemos una especie de input vacío que a la izquierda tiene una opción preseleccionada 'GET'.

Es en ese espacio dónde tenemos que introducir la URL a la que vamos a hacer la petición, el 'GET' lo que indica es que lo que queremos es hacer una petición a dicha URL.

A la derecha tenemos un botón de 'SEND' que también es la opción que nos sale por defecto. Clicamos en este botón para enviar nuestra petición.

This screenshot shows the same Postman interface as above, but the 'Send' button has been clicked. The status bar at the bottom now displays 'Status: 200 OK Time: 112ms Size: 747 B Save Response'.

Una vez hemos clicado vemos como abajo nos aparece algo parecido a lo que vemos en la imagen de abajo. Esto es lo que hemos obtenido al hacer la petición, lo que nos ha devuelto.

This screenshot shows the response body again, identical to the one in the previous screenshot. It contains the same JSON object with the key 'value' pointing to a nested object with 'id', 'joke', and 'categories' fields.

A la derecha vemos una barra con la siguiente información :

- Status : Esta propiedad nos muestra el código de estado de la respuesta. Nos indica si la solicitud ha tenido éxito.
 - Status: 200 Ok → La solicitud ha tenido éxito.
 - Status: 400 → Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

- Status 404 → El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.
- Time : Es el tiempo que ha pasado desde que se hace la petición hasta que obtenemos la respuesta.



Si nos fijamos en lo que nos devuelve, vemos que a primera vista parece un objeto de JavaScript, pero si nos fijamos en el desplegable que tenemos por encima vemos que viene preseleccionado “JSON”.

JSON, cuyas siglas significan “JavaScript Object Notation” - “Notación de Objeto de JavaScript” es un formato de texto que nos permite convertir JavaScript en texto para el intercambio de datos.

Podemos convertir JavaScript en JSON y viceversa. Cuando tratemos datos que nos llegan desde una petición a través de un fetch, estos nos llegarán en formato texto (JSON) y deberemos convertirlo en JavaScript para poder manejar y utilizar estos datos.

The screenshot shows a JSON viewer interface. At the top, there are tabs for "Body", "Cookies (1)", "Headers (15)", and "Test Results". Below that is a toolbar with "Pretty", "Raw", "Preview", "Visualize BETA", "JSON", and a copy icon. The main area displays a JSON object with the following structure:

```

1  {
2    "type": "success",
3    "value": [
4      {
5        "id": 172,
6        "joke": "Some kids play Kick the can. Chuck Norris played Kick the keg.",
7        "categories": []
8      }
]
  
```

Podemos enviar un objeto o recibir un objeto, por ejemplo, vamos a crear un objeto persona. Persona es un objeto que hemos creado en JavaScript. Si quisieramos enviar este objeto primero deberíamos convertirlo a formato texto, es decir, a JSON.

JSON.stringify()

A través de la función `stringify()`, podemos pasar un objeto javascript a JSON, simplemente tenemos que pasarle por parámetro el objeto que queremos convertir.

```
>> let persona = { nombre: 'Tintín', edad: 18, mascota: { animal: 'perro', nombre: 'Milú'}};
< undefined
>> persona
< > Object { nombre: "Tintín", edad: 18, mascota: {...} }
>> JSON.stringify(persona)
< "{\"nombre\":\"Tintín\", \"edad\":18, \"mascota\":{\"animal\":\"perro\", \"nombre\":\"Milú\"}}"
>>
```

JSON.parse()

Para pasar un objeto en formato JSON a un objeto de JavaScript utilizamos `JSON.parse()`.

```
>> personaEnTexto
< "{\"nombre\":\"Tintín\", \"edad\":18, \"mascota\":{\"animal\":\"perro\", \"nombre\":\"Milú\"}}"
>> JSON.parse(personaEnTexto)
< > Object { nombre: "Tintín", edad: 18, mascota: {...} }
```

```
// JSON – JavaScript Object Notation
// convierte javascript en texto para poder enviarlo por internet
let persona = {
  nombre: 'Tintín',
  edad: 18,
  mascota: {
    animal: 'perro',
    nombre: 'Milú'
  }
};

let personaEnTexto = JSON.stringify(persona);
console.log(personaEnTexto);
let personaCopia = JSON.parse(personaEnTexto);
console.log(personaCopia);
```

👉 Lo convertimos en texto para enviarlo. Cuando pedimos información a una página de internet esta información nos llega en formato texto y queremos parsearlo a Objeto javascript para poder manejarlo e incluirlo en nuestra página.

setTimeout(función, tiempo)

setTimeout() es una función que nos permite retrasar la ejecución de una función x milisegundos. Recibe dos parámetros, una función a ejecutar y el número de milisegundos que queremos que transcurran hasta que se ejecute esa función, de esta manera establece un temporizador y ejecuta una función después de que el temporizador expire.

```
function despedida() {
    console.log('adiós');
}

setTimeout(function holaOtraVez() {
    console.log('hola otra vez');
}, 2000); // setTimeout recibe dos parametros: una función a ejecutar y el número de milisegundos después del que se le llama
```

La función la podemos definir dentro de los paréntesis de setTimeout, o fuera. Si la definimos fuera hay que tener en cuenta que no debemos utilizar los paréntesis, solamente utilizamos el nombre de la función específica a la que nos referimos, de esta manera indicamos que cuando pasen x milisegundos busqué una función con ese nombre y la ejecute.

```
function despedida() {
    console.log('adiós');
}

function holaOtraVez() {
    console.log('hola otra vez');
}

// setTimeout recibe dos parametros: una función a ejecutar y el número de milisegundos después del que se le llama
setTimeout(holaOtraVez, 5000);
```

Ejemplo práctico

The screenshot shows the Postman application interface. At the top, it says "Untitled Request". Below that, there's a header bar with "GET" selected, the URL "http://api.icndb.com/jokes/random", and buttons for "Send" and "Save". Under the URL, there are tabs for "Params", "Authorization", "Headers (8)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is active, showing a table for "Query Params" with one entry: "Key" (Value) and "Value" (Description). Below the table, there are tabs for "Body" (selected), "Cookies (1)", "Headers (14)", and "Test Results". The "Body" tab shows the response in "Pretty" format, which is a JSON object:

```
1 {  
2     "type": "success",  
3     "value": {  
4         "id": 330,  
5         "joke": "President Roosevelt once rode his horse 100 miles. Chuck Norris carried his the same distance in half the time.",  
6         "categories": []  
7     }  
8 }
```

At the bottom right of the response area, it says "Status: 200 OK Time: 87ms Size: 642 B Save Response".

Vamos a hacer una petición a la API de chuck norris, para que nos devuelva un chiste aleatorio.

Al hacer la petición desde nuestro programa, nos vamos a encontrar con dos problemas, uno el formato en el que nos va a llegar la respuesta (JSON) y el segundo es que no sabemos cuánto tiempo va a tardar en llegarnos la respuesta.

Veamos un ejemplo con setTimeout:

```
<script>
    console.log('uno');

    function mostrarDos() {
        console.log('dos');
    }
    setTimeout(mostrarDos, 2000);

    console.log('tres');
    console.log('cuatro');
    console.log('cinco');
</script>
```

```
uno
tres
cuatro
cinco
dos
```

En este caso estamos mostrando por consola los números del uno al cinco en orden pero al retrasar la instrucción que muestra el número dos, los siguientes se muestran antes. Este mismo problema vamos a tener cuando hagamos una petición, porque el código que tengamos por debajo de la petición se va a ejecutar antes de que nos llegue la respuesta. Para abordar este problema utilizamos una función parecida a `setTimeout()` para evitarlo, ¿Por qué no `setTimeout()`? Porque no sabemos **específicamente cuánto tiempo** va a tardar en llegar la respuesta.

fetch()

`fetch()` es un método al cual le pasamos un string como parámetro y nos sirve para indicar a nuestro programa que vamos a hacer una petición de datos. El string que le pasamos será la URL que nos proporciona la API, la misma que utilizamos en postman.

```
<script>
    fetch('http://api.icndb.com/jokes/random')
</script>
```

Cómo no sabemos cuánto va a tardar en llegar la respuesta utilizamos el método `.then()` después de la petición que hayamos definido. Este método va a recibir una función que a su vez recibe como primer parámetro la respuesta de internet, el código que contiene entre llaves dicha función SÓLO se va a ejecutar cuando haya llegado la respuesta. Por lo tanto,

`.then()` nos sirve para indicar que el código debe ejecutarse únicamente cuando lleguen los datos.

```
<script>
  fetch('http://api.icndb.com/jokes/random')
    .then(function cogerRespuesta(respuesta) {
      console.log(respuesta);
    });

</script>
```

Una vez nos llega la respuesta, debemos recordar que esta nos llega en formato texto (JSON), además nosotros queremos acceder al 'BODY' que si nos fijamos es lo que nos muestra el postman por defecto cuando hacemos una petición.

The screenshot shows the Postman interface with the 'Body' tab selected. The response body is displayed in a pretty-printed JSON format:

```
1  {
2   | "type": "success",
3   | "value": {
4   |   "id": 377,
5   |   "joke": "When Arnold says &quot;I'll be back&quot; in Terminator movie it is implied that he's going to ask Chuck Norris for help.",
6   |   "categories": []
7   }
8 }
```

Podemos hacer las dos cosas a la vez, a partir de la respuesta acceder al body y pasarlo a formato javascript a partir de un método `.json()`.

Pero aun así, si hacemos un `console.log()` de la `respuesta.json()`, vemos como sigue siendo una promesa , ¿Por qué? Porque el proceso de obtener el body y pasarlo a javascript también va a llevar un tiempo, por eso utilizaremos otro `then()`.

```
<script>
  fetch('http://api.icndb.com/jokes/random')
    .then(function cogerRespuesta(respuesta) {
      console.log(respuesta);
    });

</script>
```

Utilizaremos otro `.then()` para indicar que cuando termine de parsear los datos se ejecute el bloque de código que tenemos en los corchetes pero, ¿Cómo nos llegan estos datos? A través de una función, que podemos llamar como queramos, y que va a recibir como

parámetro la información que hemos recibido de internet ya en formato javascript, con lo que ya podremos manejar y utilizar dicha información.

```
<script>
    fetch('http://api.icndb.com/jokes/random')
        .then(function cogerRespuesta(respuesta) {
            // .json() es equivalente a obtener el body y hacer un JSON.parse() del mismo
            return respuesta.json();
        })
        .then(function cogerDatos(datos) {
            console.log(datos);
        });
</script>
```

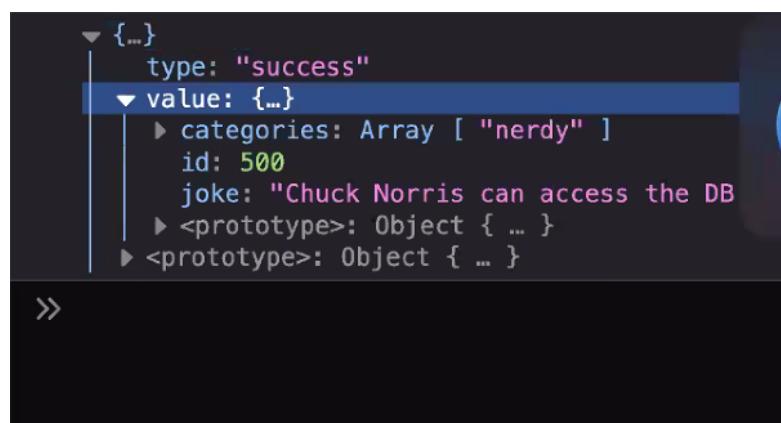
Resumen:

Utilizamos el método `fetch()` para hacer una petición a internet, para ello pasaremos por parámetro al método la url en formato string.

Como no sabemos cuánto tiempo va a pasar hasta que recibamos la respuesta utilizamos el método `.then()`, este método recibe una función que, a su vez, recibe la respuesta, y una vez ha llegado dicha respuesta se ejecuta el código que encontramos entre los corchetes. En este caso devolvemos la respuesta utilizando el método `.json()`, que obtiene a partir de la respuesta el body y lo pasa a javascript.

Como la respuesta la tenemos que pasar a javascript y esto tampoco sabemos cuánto va a tardar, lo que hacemos es utilizar otro `.then()` al que pasamos una función que recibirá estos datos ya parseados y cuando los reciba se ejecutará el código que definimos entre corchetes.

Si mostramos por consola el valor de 'datos', el parámetro que recibe la función 'cogerDatos', vemos cómo lo que recibimos es un objeto, un objeto de javascript cuya información se corresponde con la que podíamos observar a través de postman.



```
{  
    "type": "success",  
    "value": {  
        "id": 377,  
        "joke": "When Arnold says \"I'll be back\" in Terminator movie it is implied that he's going to ask Chuck Norris for help.",  
        "categories": []  
    }  
}
```

Cómo es un objeto de javascript, si por ejemplo queremos mostrar el chiste 'joke' que nos devuelve, tendremos que acceder al objeto 'datos' a la propiedad 'value' que, a su vez es otro objeto, y por último acceder a la propiedad 'joke'.

```
fetch('http://api.icndb.com/jokes/random')  
    .then(function cogerRespuesta(respuesta) {  
        // .json() es equivalente a obtener el body y hacer un JSON.parse() del mismo  
        return respuesta.json();  
    })  
    .then(function cogerDatos(datos) {  
        let chiste = datos.value.joke;  
        console.log(chiste);  
    });
```

También podemos introducir este chiste como contenido de un párrafo para mostrarlo en el html

```
<body>  
    <p id="chiste"></p>  
    <script>  
  
        fetch('http://api.icndb.com/jokes/random')  
            .then(  
                function cogerRespuesta(respuesta) {  
                    // .json() es equivalente a obtener el body y hacer un JSON.  
                    return respuesta.json();  
                }  
            )  
            .then(  
                function cogerDatos(datos) {  
                    let chiste = datos.value.joke;  
                    document.getElementById('chiste').innerHTML = chiste;  
                }  
            );  
  
    </script>
```