

Comunicación y funcionalidades de los navegadores

"La Red o Internet es considerada un gran aliado a la hora de potenciar la creatividad. Además utilizamos la Red para acceder a la información, comunicar, almacenar datos y divertirnos."

Anthony Peter Buzan – Psicólogo y escritor, inventor del Mind Mapping



Photo by [Aditya Chinchure](#) on [Unsplash](#)

El surgimiento de la computadora e Internet cambiaron la forma en la que nos comunicamos –y muchas otras cosas más– y todo se volvió más veloz. Actualmente contamos con una gran variedad y cantidad de canales para poder conectarnos entre nosotros/as y con diferentes lenguajes para hacerlo.

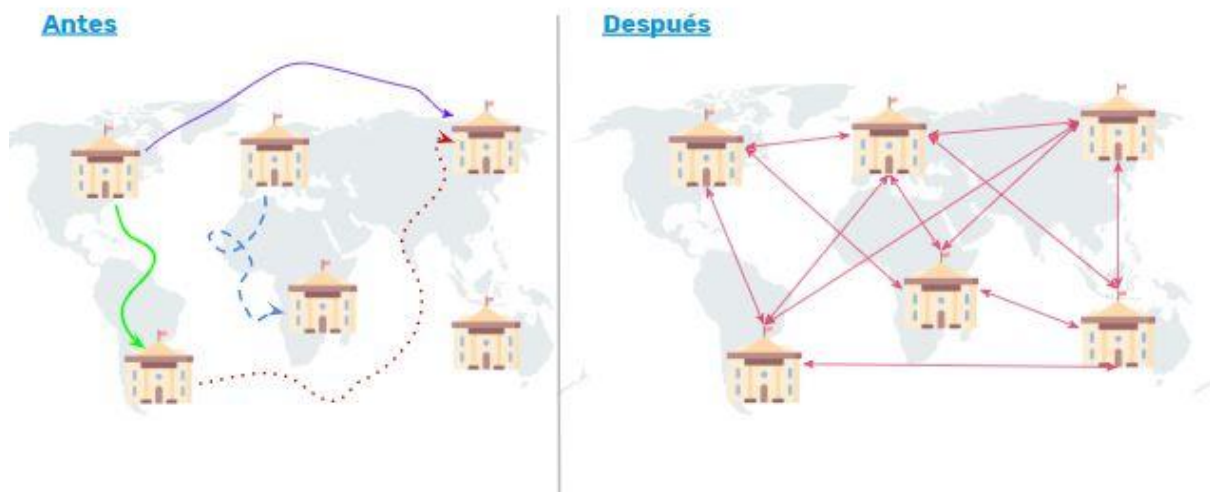
Es aquí donde vuelve a tomar importancia el backend en JavaScript para comprender el funcionamiento de lo que se encuentra detrás de una computadora y, en definitiva, lo que permite esa versatilidad y velocidad que



todos experimentamos como usuarios/as. En esta bitácora vamos a profundizar qué sucede en el servidor, en el navegador y en las bases de datos. También vamos a articular con los contenidos de las anteriores bitácoras, desde las promesas y los objetos hasta las funciones asíncronas.

Protocolos de comunicación entre computadoras

Para comprender los contenidos que veremos, vamos a recordar algo que hemos visto en el primer Bloque sobre la invención de internet y cómo se comunican las computadoras entre sí:



Vimos que en un primer momento existió una multiplicidad de protocolos cuando se masificaron las redes. Luego que fue necesario estandarizar la comunicación para que todas las computadoras pudieran comunicarse y, para ello, se creó un [protocolo](#), el idioma universal de las computadoras. Existen diferentes tipos de protocolos según el mecanismo que utilice y el tipo de información que recibe. Por ejemplo:

- Telnet: acceso remoto
- SMTP: correo electrónico
- FTP: transmisión de archivo
- HTTP: comunicación de redes



- HTTPS: comunicación de redes segura

Vamos a centrarnos en los **protocolos HTTP** y **HTTPS** que nos permiten compartir información entre computadoras como:



La comunicación ocurre entre dos partes, el **cliente** y el **servidor**, que piden y envían información respectivamente. Cuando desde el cliente solicitamos que el servidor realice una acción, debemos enviar los datos necesarios para el procesamiento y el tipo de acción a realizar.

Las acciones más comunes son:

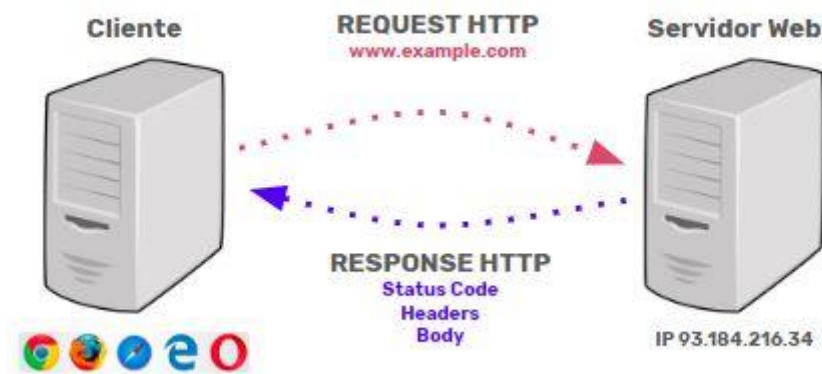
- **GET** - De lectura. Le solicitamos algo al servidor y la respuesta devolverá la información pedida. *Ej: Pedir el avatar de usuario*
- **POST** - De creación. Le solicitamos al servidor que cree almacene los datos que le estamos enviando. *Ej: Crear un nuevo usuario*
- **PUT** - De actualización o reemplazo. Le solicitamos al servidor que actualice un dato. *Ej: Un usuario actualiza su contraseña.*
- **DELETE** - De eliminación. Le solicitamos al servidor que elimine algo. *Ej: Borra una foto*

Por ejemplo si deseamos crear un usuario en una aplicación, debemos tener en cuenta estos 3 datos.

- URL: la dirección donde el servidor estará escuchando el llamado del cliente para procesar acción. *Ej: https://servidor.com/crear_usuario*
- Tipo de acción: GET, POST, PUT o DELETE
- Parámetros: la información que debo enviar al servidor para que la procese. *Ej: Los datos del usuarios*



¿Qué sucede cuando accedemos a un sitio web?



Cada documento tiene una **URL** (Uniform Resource Locator), de la que podemos identificar tres partes. Comienza con el protocolo HTTP o HTTPS, continúa con el servidor al que le estamos pidiendo el documento y al final tenemos la ruta del documento específico que queremos acceder:

`https://servidor.com/ruta.html`

Cada servidor, computadora, tablet, smart TV o cualquier otro dispositivo que esté conectado a internet tiene una dirección **IP**. Se trata del identificador que está compuesto por una cadena de números separados en secciones por puntos. Como no son fáciles de recordar, se crearon los **dominios**, que son los nombres con el que conocemos a los sitios webs, como por ejemplo `google.com`, `amazon.com`, `microsoft.com` o cualquier otro. El registro de nombres es único y no puede haber 2 nombres iguales. **Al momento de registrar un nombre de dominio, debes añadir un servidor de DNS y éste será el encargado de transformar el nombre que registraste en la dirección IP.**

De esta manera no será necesario que ninguno de tus visitantes recuerden la dirección IP de tu servidor, solamente tipeando el nombre en su navegador llegarán a tu servidor que les devolverá el contenido que hayas programado.

Conocer el protocolo HTTP abre las puertas a lo que podemos hacer con nuestros sitios webs. Nos va a permitir traer información externa dinámicamente y utilizarla en los programas que desarrollemos.



¿Cómo traemos información?

Una de las prácticas más frecuentes en las aplicaciones web hoy en día es la de consumir [APIs](#) (o [Application Programming Interface](#)): una interfaz de programación de aplicaciones. Básicamente, es el medio por el que un programa expone una porción de su contenido para que otros lo utilicen. A la hora de invocar una [API](#), hay que chequear su documentación para poder invocar sus servicios correctamente. Por ej: [API google mail](#).

Las APIs pueden ser privadas o públicas, y pueden requerir algún tipo de autenticación para poder acceder a su contenido. Tal es el caso de utilización de una [API Key](#), una clave especial que nos brindará el mismo sistema con el que debemos identificarnos cada vez que realicemos un pedido. Existe una gran variedad de [APIs públicas](#) que pueden utilizarse abiertamente. También se puede acceder a una extensa lista de APIs web que ofrecen diversas posibilidades, donde encontraremos por ejemplo la [API de almacenamiento](#) que contiene localStorage.

Más adelante veremos cómo crear nuestras propias APIs de acuerdo a la acción que queramos realizar. Puede que otras aplicaciones utilicen nuestras APIs.

Nuestro trabajo como desarrolladores/as es crear interfaces que se comuniquen con las API para mostrar la información en la que nuestros usuarios la esperan, como hemos estado trabajando en los encuentros anteriores, JavaScript nos provee la función [fetch](#) para realizar llamadas a URLs externas.

Recordemos que `fetch` usa [promesas](#). Recibe los argumentos necesarios para indicar el tipo y contenido del request, y maneja la respuesta del servidor resolviendola a un objeto Response a través de promesas. Entre sus funciones se encuentran enviar datos y el chequeo de respuestas.

Como hemos visto recientemente, para comunicarnos con una API necesitamos tres datos fundamentales: URL, tipo de acción y parámetros. En el caso de `fetch`, el único obligatorio es el primero —la URL— con la cual nos queremos comunicar; el segundo, si no lo enviamos de manera predeterminada, toma GET (pedir información al servidor); el tercero —los parámetros— no son obligatorios.



Nueva forma de trabajar las promesas

En la bitácora anterior nos detuvimos en las promesas. Nos tomamos nuestro tiempo para comprender su funcionamiento, para trabajar los procesos asíncronos, y manejar flujos de datos más complejos de una forma más sencilla y práctica. En esta bitácora vamos a expandir lo que en programación podemos llegar a hacer con las promesas.

A medida que pasa el tiempo, JavaScript va evolucionando y en su versión de 2017 fueron incorporadas las **funciones asíncronas**: son más convenientes para escribir código de una manera más simple porque permiten:

- **Escribir un código basado en promesas** como si fuese sincrónico sin bloquear el hilo principal. Devuelve una promesa y puede, en su cuerpo, especificar en que lugares queremos que trabaje de manera sincrónica.
- **Trabajar con código asíncrono complejo**, en el cual se ejecutan procesos costosos en un segundo plano sin penalizar la experiencia del usuario/a. Por ejemplo en el caso de llamadas a servidor.

Funciones asíncronas

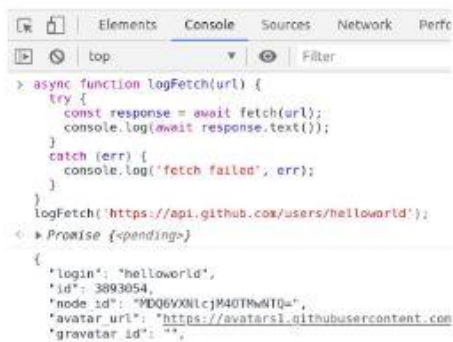
Para hacer que una función sea asíncrona se indica la palabra clave `async` antes de *function*. Cuando ésta función es llamada, devuelve una promesa. Si el cuerpo devuelve algo, esa promesa es resuelta. Si devuelve una excepción, la promesa es rechazada.

Dentro de la función `async`, podemos utilizar la palabra clave `await` antes de una expresión para esperar a una promesa en el mar antes de continuar con la ejecución de la función.

Esta nueva forma `async/await` proporciona una forma mucho más limpia, simple y legible para trabajar con promesas y de forma síncrona. En el código se puede ver claramente:



Async función

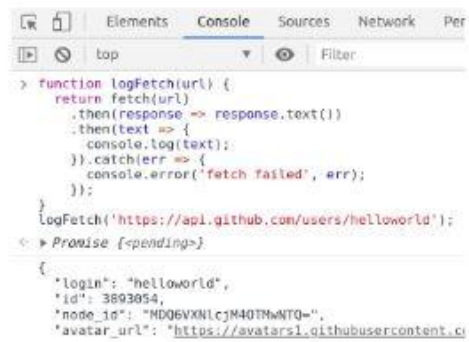


```

> async function logFetch(url) {
  try {
    const response = await fetch(url);
    console.log(await response.text());
  }
  catch (err) {
    console.log('fetch failed', err);
  }
}
logFetch('https://api.github.com/users/helloworld');
< Promise {<pending>}
{
  'login': 'helloworld',
  'id': 3893054,
  'node_id': 'MDQ6VXNlcjM4OTMwNTQ=',
  'avatar_url': 'https://avatars1.githubusercontent.com',
  'gravatar_id': '',

```

Promesa



```

> function logFetch(url) {
  return fetch(url)
    .then(response => response.text())
    .then(text => {
      console.log(text);
    }).catch(err => {
      console.error('fetch failed', err);
    });
}
logFetch('https://api.github.com/users/helloworld');
< Promise {<pending>}
{
  'login': 'helloworld',
  'id': 3893054,
  'node_id': 'MDQ6VXNlcjM4OTMwNTQ=',
  'avatar_url': 'https://avatars1.githubusercontent.com',

```

Cierre

Quedó claro que el navegador es nuestro mejor aliado en JavaScript para la comunicación mediante protocolos. También vimos cómo se comunican las aplicaciones a través de las APIs y sabemos cómo utilizarlas para el intercambio de datos. ¡Prepárate para comenzar a poner en práctica todo esto en el próximo encuentro!

¡Prepárate para el próximo encuentro!

Profundiza


 [Uso de Fetch](#)

 [¿Qué es una API?](#)

Comunidad y Challenge



Existen innumerables servicios que ofrecen una API para conectarse y que te puede proveer de información muy valiosa, algunos públicos, otros privados, pagos y gratuitos.

 Googlea cómo un/a developers y encuentra al menos 5 APIs. En el próximo encuentro nos contarás qué APIs encontraste y qué problemas resuelven. Entre todos armaremos una lista de APIs con su descripción para subir a la plataforma y que nos quede disponible para todos. Verás que hay cosas fantásticas para darle valor agregado a tus desarrollos.

