Cuestionario sobre Métodos de HTTP en JavaScript:

1. **¿Qué son los métodos de HTTP en JavaScript y cuál es su propósito principal?**
2. **Enumera al menos 5 métodos de HTTP en JavaScript y describe brevemente la función de cada uno.**
3. **¿Cuál es la diferencia entre los métodos GET y POST en una solicitud HTTP en JavaScript?**
4. **Explica cómo se puede realizar una solicitud GET en JavaScript utilizando el objeto XMLHttpRequest.**
5. **¿Qué es AJAX y cómo se relaciona con los métodos de HTTP en JavaScript?**
6. **Describe el proceso para enviar una solicitud POST en JavaScript utilizando la biblioteca Axios.**
7. **¿Por qué es importante comprender y utilizar los métodos de HTTP en el desarrollo web con JavaScript?**
8. **¿Cuál es la importancia de las respuestas de estado en las solicitudes HTTP y cómo se manejan en JavaScript?**
9. **¿Qué papel juegan las promesas en las solicitudes HTTP en JavaScript y por qué son útiles?**
10. **¿Cuál es la diferencia entre el método fetch y el objeto XMLHttpRequest en términos de realizar solicitudes HTTP en JavaScript?**

Tarea de Investigación sobre Métodos de HTTP en JavaScript:

Realiza una investigación más profunda sobre los métodos de HTTP en JavaScript y su aplicación en el desarrollo web. Incluye los siguientes puntos en tu tarea:

* **Historia y Evolución:** Explora la evolución de los métodos de HTTP en JavaScript y cómo han impactado en el desarrollo web a lo largo del tiempo.
* **Principales Métodos:** Detalla al menos 5 métodos de HTTP en JavaScript, sus características, usos comunes y ejemplos de implementación.
* **Bibliotecas y Frameworks:** Investiga las bibliotecas y frameworks populares utilizados en JavaScript para realizar solicitudes HTTP, como Axios, Fetch API, y jQuery AJAX.
* **Ventajas y Desventajas:** Analiza las ventajas y desventajas de utilizar diferentes métodos de HTTP en JavaScript, considerando aspectos como rendimiento, facilidad de uso y mantenimiento del código.
* **Casos de Uso:** Proporciona ejemplos de casos de uso reales donde los métodos de HTTP en JavaScript son fundamentales para la interacción entre el cliente y el servidor en aplicaciones web.
* **Tendencias Futuras:** Comenta sobre las tendencias futuras en el uso de métodos de HTTP en JavaScript y cómo se espera que evolucionen para satisfacer las demandas del desarrollo web moderno.

Esta tarea te permitirá profundizar en el conocimiento de los métodos de HTTP en JavaScript y su importancia en el desarrollo web actual. ¡Espero que esta guía te sea útil para completar tu cuestionario y tarea de investigación!

La interacción entre el [frontend y el backend](https://kinsta.com/es/blog/backend-vs-frontend/) de los sitios web es posible a través de las peticiones HTTP. Funcionalidades como la actualización de un usuario nuevo/antiguo en una base de datos, la publicación de una entrada en el blog desde el [panel de control de WordPress](https://kinsta.com/es/base-de-conocimiento/panel-control-administracion-wordpress/), la eliminación de una foto de tu perfil y la actualización de tu biografía requieren una interacción entre un servidor y un cliente para modificar los datos.

Cuando construimos aplicaciones web — tanto frontend como full stack web apps — a menudo interactuamos con datos almacenados en diferentes bases de datos y servidores. Estos servidores pueden pertenecer a terceros o ser creados por los desarrolladores de un proyecto.

Para interactuar con estos servidores, podemos hacer uso de diferentes métodos HTTP para solicitar datos. Podemos crear, leer, actualizar y eliminar (o CRUD) datos en los servidores utilizando verbos HTTP específicos como POST, GET, PUT/PATCH y DELETE

Las [peticiones HTTP](https://kinsta.com/es/base-de-conocimiento/que-es-una-peticion-http/) en JavaScript son un conjunto de métodos predefinidos que se utilizan para interactuar con los datos almacenados en los servidores.

Las [peticiones HTTP](https://kinsta.com/es/base-de-conocimiento/que-es-una-peticion-http/) en JavaScript son un conjunto de métodos predefinidos que se utilizan para interactuar con los datos almacenados en los servidores.

2.

* Son los siguiente: **GET:** Es utilizado únicamente para **consultar información** al servidor, muy parecidos a realizar un SELECT a la base de datos. No soporta el envío del payload
* **POST:** Es utilizado para solicitar la **creación de un nuevo registro**, es decir, algo que no existía previamente, es decir, es equivalente a realizar un INSERT en la base de datos. Soporta el envío del payload.
* **PUT**: Se utiliza para **actualizar por completo un registro existente**, es decir, es parecido a realizar un UPDATE a la base de datos. Soporta el envío del payload.
* **PATCH**: Este método es similar al método PUT, pues permite actualizar un registro existente, sin embargo, este se utiliza cuando **actualizar solo un fragmento del registro** y no en su totalidad, es equivalente a realizar un UPDATE a la base de datos. Soporta el envío del payload
* **DELETE**: Este método se utiliza para **eliminar un registro existente**, es similar a DELETE a la base de datos. No soporta el envío del payload.
* **HEAD**: Este método se utilizar para **obtener información sobre un determinado recurso** sin retornar el registro. Este método se utiliza a menudo para probar la validez de los enlaces de hipertexto, la accesibilidad y las modificaciones recientes.

3.

* Son los siguiente: **GET:** Es utilizado únicamente para **consultar información** al servidor, muy parecidos a realizar un SELECT a la base de datos. No soporta el envío del payload
* **POST:** Es utilizado para solicitar la **creación de un nuevo registro**, es decir, algo que no existía previamente, es decir, es equivalente a realizar un INSERT en la base de datos. Soporta el envío del payload.

4.

Utilizas la petición GET cuando quieres recuperar datos de un servidor. Para enviar una solicitud GET con éxito utilizando XMLHttpRequest en JavaScript, debes asegurarte de que lo siguiente se hace correctamente:

1. Crea un nuevo objeto XMLHttpRequest.
2. Abre una conexión especificando el tipo de petición y el punto final (la URL del servidor).
3. Envía la petición.
4. Espera la respuesta del servidor.
5. Lo primero que hicimos fue crear un nuevo objeto XMLHttpRequest y almacenarlo en una variable llamada xhr. Esto es:
6. const xhr = new XMLHttpRequest();
7. A continuación, especificamos el tipo de petición (GET) y el punto final al que se enviará la petición (en este caso,»<https://jsonplaceholder.typicode.com/users>«)ː
8. xhr.open("GET", "https://jsonplaceholder.typicode.com/users");
9. Para enviar la petición al servidor, utilizamos el método send().
10. Cuando el servidor devuelve los datos, puedes especificar el formato en el que se devuelven los datos.
11. En la mayoría de los casos, se utiliza JSON. Así que nos aseguramos de que nuestros datos se devuelvan en JSON añadiendo esto:
12. xhr.responseType = "json";
13. En este punto, hemos enviado con éxito una petición GET.

POST

El ejemplo de código que verás en esta sección es similar al de la sección anterior. La principal diferencia es que las peticiones POST requieren cierta información que suele almacenarse en un objeto antes de ser enviada al servidor.

Aquí tienes un ejemplo:

const xhr = new XMLHttpRequest();

xhr.open("POST", "https://jsonplaceholder.typicode.com/posts");

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

const body = JSON.stringify({

title: "Hello World",

body: "My POST request",

userId: 900,

});

xhr.onload = () => {

if (xhr.readyState == 4 && xhr.status == 201) {

console.log(JSON.parse(xhr.responseText));

} else {

console.log(`Error: ${xhr.status}`);

}

};

xhr.send(body);

En el código anterior, la información que se enviará al servidor se almacena en una variable llamada body. Contiene tres propiedadesː title, body, y userId.

Ten en cuenta que la variable body que contiene el objeto debe ser convertida en un objeto JSON antes de ser enviada al servidor. La conversión se realiza mediante el método JSON.stringify().

Para asegurarte de que el objeto JSON se envía al servidor, se pasa como parámetro al método send():

xhr.send(body);

5. XMLHttpRequest es un objeto incorporado en JavaScript que se utiliza para interactuar con los servidores y cargar contenido en las páginas web sin recargar el navegador.

6.

#### Cómo enviar una solicitud POST en JavaScript con Axios

La petición POST en Axios toma dos parámetros: la URL y los datos a enviar al servidor. Puedes almacenar los datos en una variable o pasarlos directamente como parámetro.

A continuación te explicamos cómo hacerlo:

axios.post("https://jsonplaceholder.typicode.com/posts", {

title: "POST request with Axios",

body: "POST request",

userId: 10,

})

.then(function (response) {

console.log(response.data);

})

.then((error) => console.log(error))

En el ejemplo anterior, estamos enviando datos al servidor. Los datos se pasan como segundo parámetro al método post().

Si la solicitud se envía con éxito, verás el resultado registrado en la consola.

8. Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases:

1. Respuestas informativas (100–199),
2. Respuestas satisfactorias (200–299),
3. Redirecciones (300–399),
4. Errores de los clientes (400–499),
5. y errores de los servidores (500–599).

9.

Las promesas en JavaScript sirven para manejar operaciones asíncronas de manera más estructurada y legible. Cuando realizas tareas que no se completan de inmediato, como llamadas a API, lecturas de archivos o consultas a bases de datos, **JavaScript necesita una forma de gestionar estas operaciones sin bloquear la ejecución del código**.

Las promesas cumplen varias funciones clave:

### **Manejo del flujo asíncrono óptimo**

Te permiten escribir un código más limpio y comprensible al evitar el anidamiento excesivo de callbacks. En lugar de anidar múltiples callbacks, puedes encadenar métodos**.then()**de manera más legible.

### **Gestión de errores**

Brindan un mecanismo más efectivo para manejar errores en operaciones asíncronas. Puedes usar el método**.catch()**para capturar errores que ocurren en cualquier parte de la cadena de promesas.

### **Encadenamiento de operaciones**

Permiten encadenar múltiples operaciones asíncronas de una manera más intuitiva. El resultado de una promesa puede ser pasado como entrada a la siguiente promesa en la cadena utilizando el método .**then()**.

### **Manejo de paralelismo**

Puedes utilizar constructores como **Promise.all()** y **Promise.race()**para manejar múltiples promesas al mismo tiempo. El primero espera a que todas las promesas se cumplan antes de continuar, mientras que el segundo se resuelve tan pronto como una de las promesas se cumple.

### **Legibilidad y mantenibilidad**

Las promesas hacen que el código asíncrono sea más fácil de leer y mantener, ya que proporcionan una estructura clara para manejar resultados exitosos y errores, en lugar de tener que rastrear flujos de control complejos en callbacks anidados.

### **Compatibilidad con async/await**

Asimismo, forman la base de las funciones **async** y **await,** una característica introducida en versiones más recientes de JavaScript. **async/await**simplifica aún más el manejo de código asíncrono al hacer que parezca código síncrono, mejorando la legibilidad.

Pendiente cumplida y rechazada son los tres posibles estados de la promesa.

10.   
Hay una diferencia importante entre el objeto de respuesta en XMLHttpRequest y Fetch. XMLHttpRequest devuelve los datos como respuesta, mientras que el objeto de respuesta de Fetch contiene información sobre el objeto de respuesta en sí mismo.