Protocolo HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo de comunicación que permite la transferencia de datos en la web. Fue desarrollado por Tim Berners-Lee en 1989 y es el fundamento de la World Wide Web (WWW). HTTP define cómo los mensajes se formatean y transmiten, y cómo los servidores y navegadores deben actuar en respuesta a varios comandos. Este protocolo se utiliza para recuperar recursos, como documentos HTML, imágenes, videos, etc., desde un servidor web.

Funciones Principales del Protocolo HTTP

- **Transferencia de Hipertexto:** HTTP permite la transferencia de documentos HTML que contienen hipervínculos, lo que facilita la navegación entre diferentes páginas web.
- **Interacción Cliente-Servidor:** HTTP sigue un modelo de cliente-servidor, donde el cliente (usualmente un navegador web) envía solicitudes al servidor, y este responde con los datos solicitados.
- **Método de Solicitud-Respuesta:** HTTP se basa en un modelo de solicitud-respuesta, donde el cliente envía una solicitud HTTP y el servidor responde con un mensaje que incluye la información solicitada o un mensaje de error.
- **Estateless:** HTTP es un protocolo sin estado, lo que significa que cada solicitud es independiente y no se guarda información sobre solicitudes anteriores.
- **Caché:** HTTP permite la implementación de mecanismos de caché para almacenar respuestas de manera temporal, mejorando la eficiencia de la red y reduciendo la carga del servidor.
- **Autenticación y Seguridad:** HTTP puede trabajar con mecanismos de autenticación y se puede asegurar mediante HTTPS (HTTP Secure), que añade cifrado a las comunicaciones utilizando SSL/TLS.

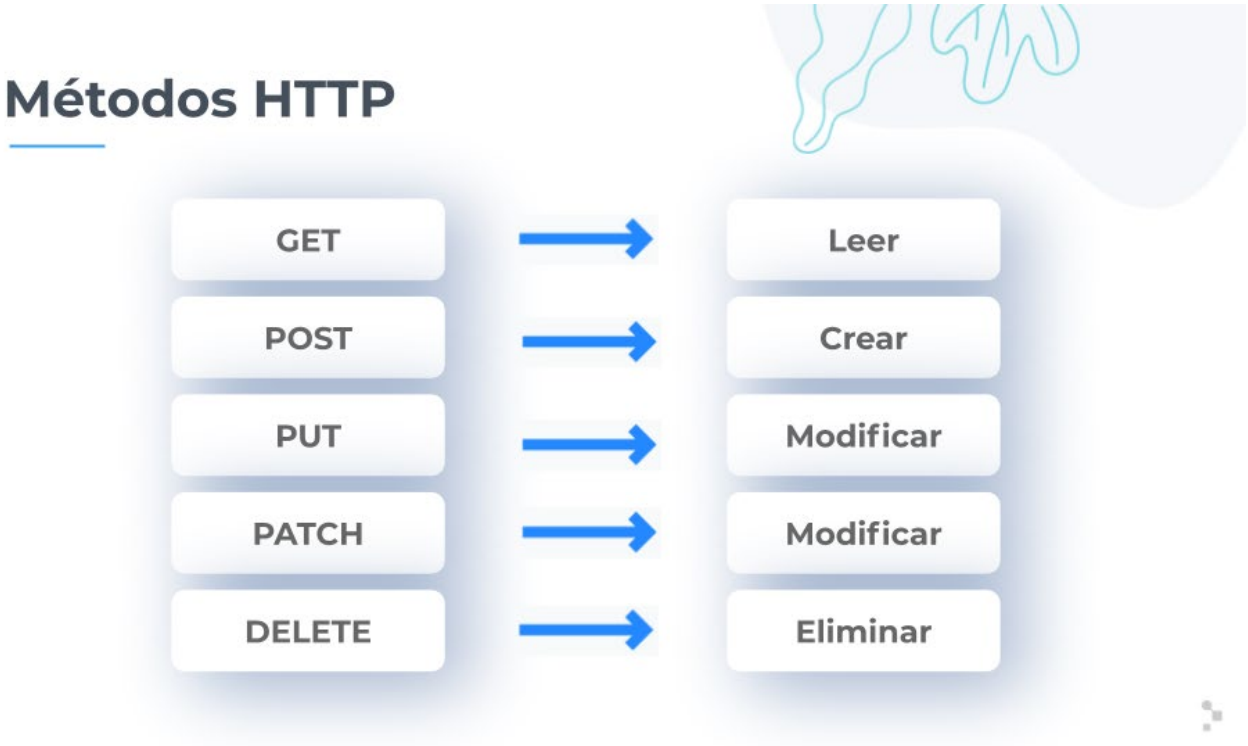
Características del Protocolo HTTP

- **Simplicidad:** HTTP es un protocolo simple y fácil de implementar. Utiliza una estructura de texto clara y comprensible.
- **Extensible:** Puede ser extendido mediante cabezales HTTP adicionales o métodos personalizados.
- **Flexible:** Permite la transferencia de una amplia gama de datos, no solo HTML, sino también JSON, XML, imágenes, videos, etc.
- **Estateless:** No guarda información sobre conexiones anteriores, lo que simplifica la gestión de la comunicación.
- **Conexión sobre TCP/IP:** Funciona sobre el protocolo de transporte TCP, garantizando una entrega confiable de los datos.

Métodos HTTP

Los métodos HTTP definen la acción que se quiere realizar en el servidor. A continuación, se describen los métodos más comunes:

- **GET:** Se utiliza para solicitar datos de un servidor. Es el método más común y se emplea para recuperar recursos como páginas web, imágenes, etc. Es un método idempotente y seguro, lo que significa que realizar varias veces la misma solicitud no debe cambiar el estado del servidor.
- **POST:** Se utiliza para enviar datos al servidor, como en el caso de formularios web. Es un método no idempotente, lo que significa que realizar la misma solicitud varias veces puede tener efectos secundarios.
- **PUT:** Se utiliza para actualizar o reemplazar un recurso existente en el servidor. Si el recurso no existe, el servidor puede crearlo. Es idempotente.
- **DELETE:** Se utiliza para eliminar un recurso específico en el servidor. Es idempotente.
- **HEAD:** Similar a GET, pero solo solicita los encabezados HTTP y no el cuerpo del mensaje. Se utiliza comúnmente para verificar si un recurso existe o para obtener metadatos.
- **OPTIONS:** Permite a un cliente determinar las opciones de comunicación o los métodos soportados por el servidor para un recurso específico.
- **PATCH:** Similar a PUT, pero se utiliza para aplicar modificaciones parciales a un recurso en lugar de reemplazarlo por completo.
- **TRACE:** Se utiliza para realizar una prueba de bucle inverso, enviando al cliente lo que el servidor recibe. Es útil para propósitos de diagnóstico.
- **CONNECT:** Se utiliza para establecer un túnel de conexión, por ejemplo, para conexiones HTTPS a través de un proxy.



Desarrollo:

Se va a realizar un servidor HTTP que busque realizar las operaciones de tipo GET y POST, para ello se debe de tener instalado Node js.

Primero se va a crear un archivo servidor_http.js y utilizar las bibliotecas que vienen por defecto al descargar Node js.

El servidor será capaz de manejar solicitudes HTTP tanto de tipo GET como POST, respondiendo de manera específica según la ruta y el método utilizados por el cliente.

Primero, se importan dos módulos esenciales de Node.js: http y url. El módulo http es fundamental para crear servidores web en Node.js, permitiendo manejar solicitudes y enviar respuestas. Por su parte, el módulo url facilita el análisis y manejo de URLs, permitiendo extraer fácilmente la ruta y los parámetros de la solicitud.

El servidor HTTP se crea utilizando la función http.createServer(), la cual recibe como parámetro un callback que se ejecuta cada vez que llega una solicitud al servidor. Este callback tiene dos parámetros: req (la solicitud) y res (la respuesta). Dentro de esta función, se define cómo el servidor debe manejar las diferentes solicitudes que recibe.

Dentro del callback, se utiliza el método url.parse() para analizar la URL de la solicitud (req.url). Esto devuelve un objeto que contiene la ruta (pathname) y los parámetros de consulta (query). Estos valores son esenciales para determinar cómo responderá el servidor según la ruta y los parámetros proporcionados.

El código identifica el método HTTP de la solicitud utilizando req.method.toUpperCase(), que convierte el método a mayúsculas para asegurar la compatibilidad. Luego, se compara el método y la ruta solicitada para determinar qué respuesta enviar.

Si el método es GET y la ruta es /get, el servidor extrae los parámetros de la URL y los incluye en la respuesta. La respuesta es enviada con un código de estado 200 (que indica éxito) y un cuerpo en formato JSON que incluye un mensaje y los parámetros de consulta recibidos.

Si el método es POST y la ruta es /post, el servidor procede a recolectar el cuerpo de la solicitud. Dado que las solicitudes POST pueden contener datos en el cuerpo, se utiliza el evento req.on('data') para acumular los datos entrantes. Una vez que todos los datos han sido recibidos, el evento req.on('end') se dispara, indicando que el cuerpo de la solicitud está completo. El servidor entonces responde con un código de estado 200 y un cuerpo en JSON que incluye un mensaje y los datos recibidos, parseados como un objeto JavaScript.

Si la solicitud no coincide con ninguna de las rutas definidas (/get o /post), el servidor responde con un código de estado 404, que indica que el recurso solicitado no fue encontrado. El cuerpo de la respuesta incluye un mensaje en formato JSON informando al cliente que la ruta no fue encontrada.

Una vez comprendido el funcionamiento del código se va a establecer la siguiente lógica para lograr su funcionamiento en el archivo servidor_http.js:

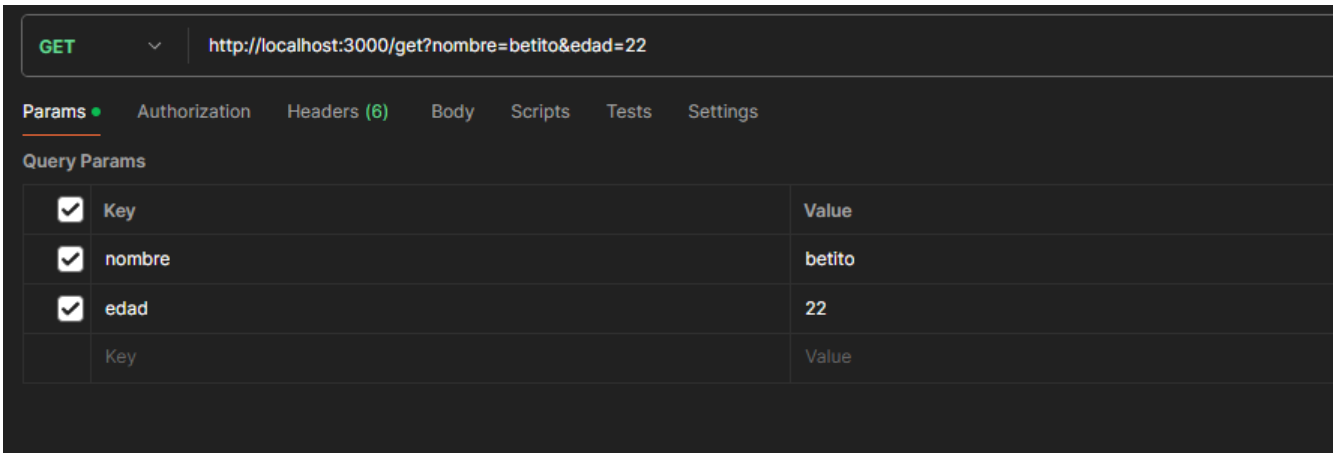
```
// Importamos los módulos necesarios
const http = require('http');
const url = require('url');

// Creamos el servidor HTTP
const server = http.createServer((req, res) => {
  // Parseamos la URL y los parámetros de la petición
  const parsedUrl = url.parse(req.url, true);
  const path = parsedUrl.pathname;
  const method = req.method.toUpperCase();

  // Configuramos las cabeceras de la respuesta
  res.setHeader('Content-Type', 'application/json');

  // Definimos las rutas para GET y POST
  if (method === 'GET' && path === '/get') {
    // Ejemplo de respuesta a una petición GET
    const queryParams = parsedUrl.query;
    res.statusCode = 200;
    res.end(JSON.stringify({
      message: 'Petición GET recibida',
      params: queryParams,
    }));
  } else if (method === 'POST' && path === '/post') {
    // Ejemplo de respuesta a una petición POST
    let body = '';

    // Recolectamos los datos del cuerpo de la petición
    req.on('data', chunk => {
      body += chunk.toString();
    });

    req.on('end', () => {
      res.statusCode = 200;
      res.end(JSON.stringify({
        message: 'Petición POST recibida',
        data: JSON.parse(body),
      }));
    });
  } else {
    // Respuesta para rutas no encontradas
    res.statusCode = 404;
    res.end(JSON.stringify({
      message: 'Ruta no encontrada',
    }));
  }
});

// Configuramos el puerto donde va a escuchar el servidor
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Servidor escuchando en el puerto ${PORT}`);
});
```

Para realizar las pruebas se requiere ejecutar el servidor con el comando:

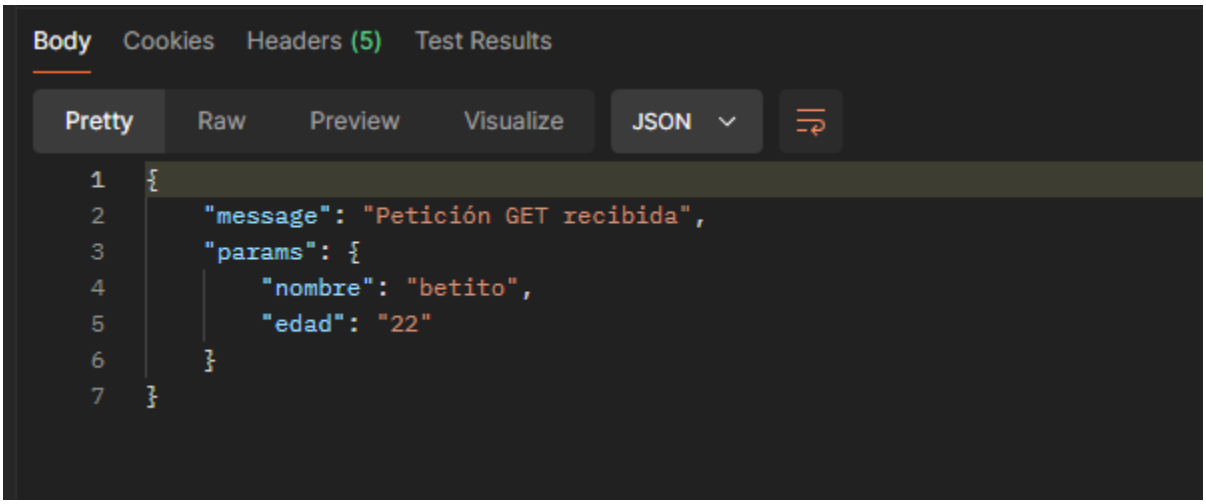
node servidor_http.js

Como pruebas se va a utilizar el navegador y POSTMAN:

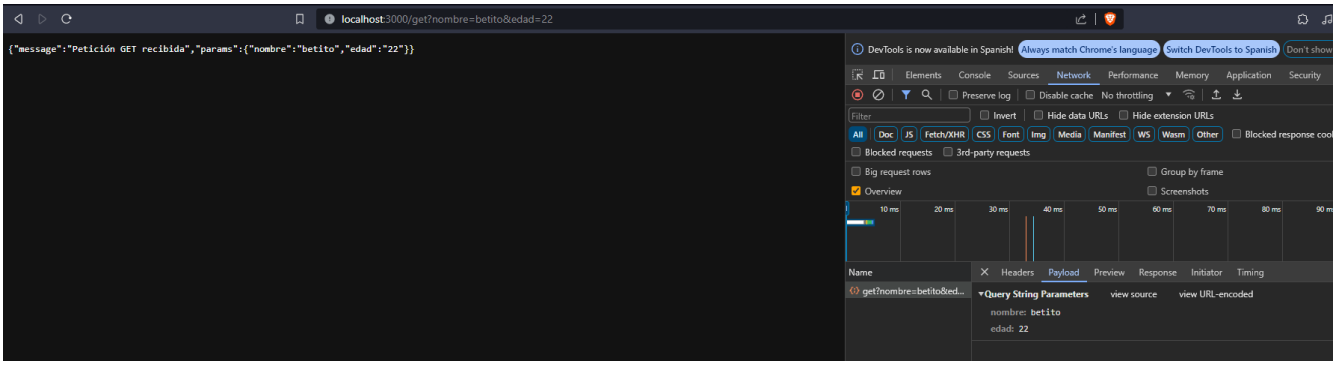
Primero para el método GET se mandan dos parámetros un nombre y una edad:



Teniendo la siguiente respuesta:



Ahora si se introduce la misma URL dentro del navegador y abrimos el inspector en el apartado de Redes o Network podremos ver la respuesta como se ve a continuación:



Para el método POST se también se manda a la URL dos parámetros: