Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.Dibujo en blanco y negro

El contenido generado por IA puede ser incorrecto.

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Aplicaciones para comunicaciones en red

Academia “Sistemas Distribuidos”

Plan 2020

Práctica 14:

“**Protocolo HTTP**”

2015090269

González González Armando Omar

Profesor: Ojeda Santillan Rodrigo

Contenido

[Objetivo 3](#_Toc201743052)

[Introducción 3](#_Toc201743053)

[Desarrollo 4](#_Toc201743054)

[3.1 Análisis del Código del Servidor (server.js) 4](#_Toc201743055)

[Conclusión 8](#_Toc201743056)

[Pregunta 8](#_Toc201743057)

[¿Cómo podrían generar un negocio a través de lo visto en la práctica? 8](#_Toc201743058)

[Bibliografía 8](#_Toc201743059)

## Objetivo

Desarrollar un servidor HTTP básico utilizando Node.js que sea capaz de manejar y responder de manera diferenciada a las solicitudes de los métodos GET y POST. El propósito es demostrar el funcionamiento práctico de estos dos métodos fundamentales del protocolo HTTP, entendiendo cómo se procesan las peticiones y se construyen las respuestas en el lado del servidor.

## Introducción

El Protocolo de Transferencia de Hipertexto (HTTP) es el protocolo de comunicación fundamental de la World Wide Web. Define un conjunto de reglas para la transferencia de recursos, como documentos HTML, imágenes y otros tipos de archivos, entre un cliente (generalmente un navegador web) y un servidor. HTTP opera sobre un modelo cliente-servidor y de solicitud-respuesta, donde el cliente envía una petición para realizar una acción y el servidor devuelve una respuesta con el resultado.

Entre sus características principales se encuentran:

* **Sin estado (Stateless):** Cada solicitud es tratada como una transacción independiente, y el servidor no guarda información de peticiones anteriores.
* **Flexible:** Permite la transferencia de diversos tipos de datos, no solo texto HTML.
* **Confiable:** Opera sobre el protocolo TCP/IP, lo que garantiza una entrega ordenada y fiable de los datos.

Los métodos HTTP definen la acción que el cliente desea realizar en el servidor. En esta práctica nos centraremos en los dos más comunes:

* **GET:** Se utiliza para solicitar y recuperar datos de un recurso específico en el servidor.
* **POST:** Se emplea para enviar datos al servidor para que sean procesados, como en el envío de un formulario web.

## Desarrollo

Para la práctica se desarrolló un servidor web utilizando Node.js. Este servidor es capaz de interpretar el método y la ruta de una solicitud HTTP para proporcionar una respuesta adecuada, ya sea para una petición GET o POST.

### **3.1 Análisis del Código del Servidor (**server.js**)**

El script del servidor implementa la lógica para manejar diferentes tipos de solicitudes HTTP.

1. **Importación de Módulos:** Se importan los módulos nativos de Node.js http y url. El módulo http es esencial para crear el servidor y manejar las comunicaciones, mientras que el módulo url se utiliza para analizar la URL de la solicitud entrante.
2. **Creación del Servidor:** Se utiliza la función http.createServer() para instanciar el servidor. Esta función recibe una función *callback* que se ejecuta cada vez que el servidor recibe una nueva petición.
3. **Procesamiento de la Petición:** Dentro del *callback*, se analiza la URL de la petición (req.url) para extraer la ruta (pathname) y el método (req.method) de la solicitud. Esto permite al servidor determinar cómo debe responder.
4. **Manejo de la Ruta GET (/get):** Si el método es GET y la ruta es /get, el servidor extrae los parámetros de la consulta (query) de la URL. Luego, envía una respuesta con código de estado 200 (éxito) y un cuerpo en formato JSON que contiene un mensaje de confirmación y los parámetros recibidos.
5. **Manejo de la Ruta POST (/post):** Si el método es POST y la ruta es /post, el servidor debe leer los datos del cuerpo de la petición. Esto se hace de forma asíncrona, escuchando el evento 'data' para acumular los trozos de datos (chunks) que llegan. Cuando todos los datos han sido recibidos, se dispara el evento 'end'. En ese momento, el servidor responde con un código 200 y un cuerpo JSON que contiene los datos recibidos.
6. **Manejo de Rutas no Encontradas:** Para cualquier otra combinación de método y ruta, el servidor responde con un código de estado 404 (No Encontrado) y un mensaje de error en formato JSON.
7. **Inicio del Servidor:** Finalmente, server.listen(PORT, ...) inicia el servidor y lo pone a escuchar en el puerto 3000.

// 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}`);

});

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 1. Ejecución de petición tipo GET en postman.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 2. Ejecución de petición tipo POST en postman.

A screenshot of a computer

AI-generated content may be incorrect.  
Figura 3. Ejecución del servidor con node.

## Conclusión

La práctica se completó con éxito, logrando implementar un servidor HTTP en Node.js capaz de diferenciar y procesar correctamente las solicitudes GET y POST. Se pudo observar de manera práctica la distinción fundamental entre ambos métodos: mientras que GET transmite datos a través de los parámetros de la URL, POST lo hace a través del cuerpo de la solicitud, lo que requiere un manejo asíncrono en el servidor para recolectar los datos.

Se concluye que el manejo de rutas y métodos HTTP es la base para la construcción de servicios web y APIs RESTful. Node.js, con sus módulos http y url, proporciona un entorno potente y eficiente para desarrollar este tipo de aplicaciones, permitiendo un control preciso sobre el flujo de comunicación entre cliente y servidor.

## Pregunta

### ¿Cómo podrían generar un negocio a través de lo visto en la práctica?

El modelo de servidor implementado es el núcleo de una API web. Un negocio podría desarrollar una API RESTful que ofrezca un servicio especializado (por ejemplo, conversión de archivos, validación de datos, procesamiento de pagos). Esta API podría ser consumida por otras aplicaciones y se monetizaría a través de un modelo de suscripción (cobrando una tarifa mensual por un número de peticiones) o un modelo de pago por uso.

## Bibliografía

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force (IETF), Jun. 1999. [En línea]. Disponible: <https://www.rfc-editor.org/rfc/rfc2616>.

[2] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 8th ed. Boston: Pearson, 2021.

[3] Node.js Foundation, "HTTP | Node.js v22.3.0 Documentation." [En línea]. Disponible: <https://nodejs.org/api/http.html>. [Consultado: 25-jun-2025].