

## **PRÁCTICA 2.1: Servidor con node.js**

José Angel Rodríguez Estrada

Juan Lopez Lopez

Tecnologías para el desarrollo de aplicaciones web

Inteligencia Artificial

## 1 Introducción

Los servidores web son pilares importantes de la infraestructura del Internet y desempeñan un papel crucial en la entrega de contenido y servicios en línea. En general un servidor web es un programa o hardware que aloja y entrega contenido web a través de Internet. Actúa como un intermediario entre los usuarios que solicitan información y los recursos que proporcionan esa información, como sitios web, aplicaciones web y archivos multimedia. El servidor web recibe las solicitudes de los clientes (navegadores web) y les proporciona las respuestas correspondientes.

## 2 Servidores web

La principal función de un servidor Web es almacenar los archivos de un sitio y emitirlos por Internet para que las páginas que aloja puedan ser visitadas por los usuarios.

Básicamente, un servidor Web es una gran computadora que guarda y transmite datos vía el sistema de redes llamado Internet. Cuando un usuario entra en una página de Internet, su navegador se comunica con el servidor, enviando y recibiendo datos que determinan qué es lo que verá en su pantalla. Por esto, podemos sintetizar el concepto: los servidores Web existen para almacenar y transmitir datos de un sitio según son solicitados por el navegador visitante.

Cada servidor Web, y cada computadora conectada a Internet, posee asignada una dirección de IP única, irrepetible, que lo identifica en la red. Tu teléfono móvil tiene una IP, al igual que tu PC o tu tablet. La dirección de IP puede pensarse como los datos del remitente en una carta postal. Cuando deseas ver un sitio Web, tu móvil, PC o tableta envía un pedido desde tu dirección de IP hacia la dirección IP del servidor que aloja los archivos del sitio en cuestión. Entonces, el servidor Web responde devolviendo los datos a esa dirección IP solicitante. Esto es lo que pasa todo el tiempo que estamos navegando sitios en Internet.

En resumen, cuando un cliente realiza una solicitud a un servidor web, se sigue un proceso para entregar la respuesta deseada[1]:

Al ingresar una URL en el navegador, la URL es en realidad el identificador de dirección del servidor web.

1. El navegador usa la URL para encontrar la dirección IP del servidor.
2. El navegador envía una solicitud HTTP de información.
3. El servidor web se comunica con un servidor de base de datos para encontrar los datos relevantes.
4. El servidor web devuelve contenido estático, como páginas HTML, imágenes, videos o archivos, en una respuesta HTTP al navegador.
5. A continuación, el navegador le mostrará la información.

Un sitio web que aloja contenido estático, como blogs, imágenes de encabezado o artículos, puede ejecutarse en un servidor web. Sin embargo, la mayoría de los sitios web y aplicaciones web son mucho más interactivos y requieren un servidor de aplicaciones.

### 3 Node.js

Node.js es un entorno de ejecución de JavaScript construido sobre el motor JavaScript V8 de Chrome, el cual toma código JavaScript y lo ejecuta directamente en el sistema operativo de la computadora. Permite a los desarrolladores ejecutar código JavaScript en el lado del servidor, a diferencia del enfoque tradicional de ejecutar JavaScript en el navegador. Node.js fue creado para abordar las limitaciones de las tecnologías tradicionales del lado del servidor, como el alto requerimiento de concurrencia, las operaciones de E/S lentas y la ejecución de código bloqueante. Su objetivo era proporcionar una plataforma altamente escalable, eficiente y capaz de manejar un gran número de conexiones concurrentes.

La arquitectura de Node.js se basa en un modelo de E/S (entrada/salida) no bloqueante y orientado a eventos, lo que la hace más eficiente y liviana para la construcción de aplicaciones intensivas en datos y en tiempo real que se ejecutan en una amplia variedad de dispositivos compartidos. A diferencia de los enfoques convencionales de servidores web, donde cada solicitud genera un nuevo subproceso que eventualmente consume toda la memoria RAM disponible, Node.js opera en un solo hilo. Esto le permite manejar miles de conexiones simultáneas gestionando eficazmente los bucles de eventos.

#### ¿Cómo funciona Node.js?

Una aplicación Node.js se ejecuta sin problemas en un único proceso. A diferencia de los programas tradicionales del lado del servidor, Node no crea hilos nuevos para cada solicitud. Por lo tanto, un servidor Node puede gestionar numerosas solicitudes concurrentes sin los problemas asociados a la concurrencia de hilos o al uso de multithreading.

Además, la arquitectura de Node.js utiliza un modelo de E/S no bloqueante y está impulsada por eventos, lo que garantiza que las aplicaciones intensivas en datos y en tiempo real se ejecuten sin problemas en una amplia gama de dispositivos compartidos. Esto hace que JavaScript sea la opción evidente para cualquier lenguaje de máquina que busque una ejecución de código ultrarrápida. Gracias a Node.js y JavaScript, los códigos operan con mayor rapidez en el entorno de servidor-cliente.

#### ¿Cómo gestiona Node.js las solicitudes simultáneas dentro de un modelo de un solo hilo?

La respuesta a las solicitudes multihilo representa una arquitectura distinta, más lenta en el bucle de eventos, y no logra manejar múltiples hilos simultáneos eficientemente.

Además, la plataforma no sigue un prototipo de respuesta/solicitud multihilo similar y sin estado. En su lugar, adopta un marco simplificado de bucle de eventos monohilo. De acuerdo con los desarrolladores de Node.js, "Libuv" es una biblioteca clave que proporciona el mecanismo de bucle de eventos. La arquitectura y el modelo de procesamiento de Node.js se basan en el modelo de eventos de JS con un mecanismo de devolución de llamadas.

#### ¿Dónde se está usando Node.js?

- [2]Aplicaciones orientadas a E/S: Node.js brilla en aplicaciones donde la entrada/salida (E/S) es un factor dominante, gracias a su naturaleza asincrónica y su capacidad para manejar múltiples operaciones simultáneas sin bloquear el hilo de ejecución.
- Aplicaciones de transmisión de datos: Su capacidad para manejar datos en tiempo real y su eficiencia en la transmisión hacen de Node.js una opción ideal para aplicaciones que necesitan transmitir datos continuamente, como transmisiones de audio o video en tiempo real.
- Aplicaciones en tiempo real y con intensidad de datos: Node.js se destaca en aplicaciones que requieren un procesamiento rápido de grandes volúmenes de datos en tiempo real, como análisis en tiempo real, seguimiento de eventos en vivo o aplicaciones de IoT.
- Aplicaciones basadas en APIs JSON: Con su capacidad nativa para trabajar con objetos JSON y su enfoque en la comunicación basada en eventos, Node.js es perfecto para construir APIs flexibles y eficientes que utilizan JSON como formato de intercambio de datos.
- Aplicaciones de página única (SPA): Node.js complementa perfectamente las aplicaciones de página única, proporcionando un backend flexible y escalable para manejar la lógica del servidor y la interacción con bases de datos, mientras que JavaScript en el lado del cliente maneja la experiencia del usuario sin necesidad de recargar la página.

## 4 Express JS

Node.js es un entorno de ejecución JavaScript altamente eficiente que se utiliza para desarrollar aplicaciones del lado del servidor. Sin embargo, aunque ofrece un rendimiento excepcional, por sí solo no proporciona funcionalidades específicas como el enrutamiento de archivos, el manejo de solicitudes HTTP y la gestión de métodos HTTP. Aquí es donde entra **Express.js**.

Express.js es un framework para Node.js diseñado para simplificar y acelerar el desarrollo de aplicaciones web y APIs. Al proporcionar una capa de abstracción sobre Node.js, Express.js simplifica tareas comunes como el enrutamiento, el manejo de solicitudes y respuestas, y la configuración de servidores. Esto hace que el desarrollo con Node.js sea más rápido y accesible, permitiendo a los desarrolladores concentrarse en la lógica de la aplicación en lugar de preocuparse por los detalles de bajo nivel.

Además de su funcionalidad básica, Express.js también ofrece una amplia gama de complementos y middleware que facilitan la integración con otras tecnologías y la ampliación de las capacidades de la aplicación. Su flexibilidad y versatilidad lo convierten en una opción popular para construir no solo aplicaciones web, sino también aplicaciones móviles multiplataforma. En resumen, Express.js complementa perfectamente Node.js al proporcionar una base sólida y herramientas poderosas para el desarrollo de aplicaciones modernas y eficientes.

Express fue concebido con el propósito de simplificar la creación de APIs y aplicaciones web, lo que resulta en un ahorro significativo de tiempo de codificación, reduciendo el esfuerzo requerido casi a la mitad. Esto se traduce en un desarrollo más rápido y eficiente, tanto para aplicaciones web como móviles.

Una de las principales ventajas de utilizar Express es que está escrito en JavaScript, un lenguaje ampliamente conocido y fácil de aprender. Esto significa que incluso aquellos que no tienen experiencia previa en programación pueden abordar el desarrollo web con relativa facilidad, lo que abre las puertas a una amplia gama de nuevos desarrolladores para adentrarse en el campo del desarrollo web.

Las razones fundamentales detrás de la creación de Express como un framework para Node.js son:

1. Eficiencia en el tiempo: Express permite un desarrollo más rápido al proporcionar una estructura clara y simplificada para construir aplicaciones web y APIs.
2. Velocidad: Gracias a su diseño liviano y su enfoque en la optimización del rendimiento, Express ofrece un rendimiento excepcionalmente rápido.
3. Costo-efectivo: Al acelerar el desarrollo y optimizar el rendimiento, Express ayuda a reducir los costos asociados con el desarrollo y mantenimiento de aplicaciones web.
4. Facilidad de aprendizaje: Su sintaxis intuitiva y su naturaleza basada en JavaScript hacen que Express sea fácil de aprender, incluso para principiantes.
5. Asincronía: Express está diseñado para manejar operaciones asincrónicas de manera eficiente, lo que lo hace ideal para construir aplicaciones web y APIs altamente escalables y receptivas.

#### 4.1 Características de Express

Desarrollo rápido del lado del servidor

- Las características de Node.js ayudan a Express a ahorrar mucho tiempo en el desarrollo de servidores.

Middleware

- El middleware es un manejador de solicitudes que tiene acceso al ciclo de solicitud-respuesta de la aplicación.

Enrutamiento

- Se refiere a cómo las URL de los puntos finales de una aplicación responden a las solicitudes del cliente.

Plantillas

- Proporciona motores de plantillas para crear contenido dinámico en páginas web mediante la creación de plantillas HTML en el servidor.

Depuración

- Express facilita la depuración al identificar la parte exacta donde se encuentran los errores. Además, ofrece herramientas y mensajes claros para ayudar a los desarrolladores a resolver problemas más rápidamente.

#### 4.2 Ventajas y limitaciones de usar Express

Utilizar Express con Node.js puede brindarnos variedad de ventajas como[2]:

1. Flexibilidad y personalización: Express es un framework "un-opinionated", lo que significa que no impone una estructura rígida de aplicación, permitiendo a los desarrolladores personalizarla según las necesidades específicas del proyecto.
2. Middleware para el manejo de peticiones: Express ofrece una amplia gama de middleware que facilita el manejo de solicitudes HTTP, permitiendo agregar funcionalidades adicionales a la aplicación de manera modular y eficiente.
3. Unificación del desarrollo: Al utilizar Node.js en el backend y en Express para manejar las solicitudes HTTP, se emplea un único lenguaje de programación (JavaScript) tanto en el frontend como en el backend, lo que simplifica el desarrollo y la coherencia del código.
4. Rendimiento optimizado con bases de datos: Express se integra fácilmente con una variedad de bases de datos como MySQL, MongoDB, entre otras, lo que permite una interacción rápida y eficiente con los datos almacenados, mejorando el rendimiento de la aplicación.

5. **Renderizado dinámico de HTML:** Express facilita el renderizado dinámico de páginas HTML mediante el paso de argumentos a las plantillas, lo que permite crear interfaces de usuario dinámicas y personalizadas de manera eficiente.

Limitaciones de Express JS:

1. **Estructura del código:** En ocasiones, la falta de una estructura clara puede hacer que el código sea difícil de entender y mantener, especialmente en proyectos grandes y complejos.
2. **Problemas con callbacks:** Aunque los callbacks son una característica fundamental de Node.js, pueden conducir a un código anidado excesivo y dificultar la comprensión del flujo de la aplicación.
3. **Claridad en los mensajes de error:** Algunos mensajes de error generados por Express pueden ser difíciles de interpretar y diagnosticar, lo que puede complicar la depuración y el mantenimiento del código.

### 4.3 Middleware

El middleware en Express.js es una pieza fundamental que permite modularizar y personalizar el flujo de manejo de las solicitudes HTTP en una aplicación. Además de ser invocado antes de que se ejecute el controlador de solicitud final, el middleware puede realizar una variedad de tareas, como autenticación, compresión de respuestas, registro de solicitudes, entre otros.

Una de las características más poderosas de Express.js es su capacidad para utilizar múltiples middleware en una cadena de procesamiento. Esto significa que puedes encadenar una serie de funciones middleware para que se ejecuten secuencialmente, cada una realizando su tarea específica antes de pasar al siguiente middleware o al controlador de solicitud final. Esta modularidad permite una gran flexibilidad en el diseño de la lógica de la aplicación, ya que puedes agregar, quitar o reorganizar middleware según sea necesario para adaptarse a los requisitos de tu aplicación.

Además de los middleware integrados proporcionados por Express.js, también puedes crear tus propios middleware personalizados para satisfacer las necesidades específicas de tu aplicación. Esto te brinda un control total sobre el flujo de solicitud y respuesta, permitiéndote implementar lógica personalizada en cualquier punto del proceso de manejo de solicitudes.

## 5 HTTP

El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo esencial para el intercambio de información en Internet. Se podría decir que es el sistema de entrega de datos en la red mundial. Funciona como los barcos que transportan mercancías a través del océano, garantizando que la información fluya de un lugar a otro de manera eficiente y segura. En el contexto de la World Wide Web, HTTP es fundamental.

HTTP es el pilar de la comunicación entre navegadores y servidores en Internet, permitiendo el intercambio de contenido como páginas web y datos. Su comprensión es crucial para quienes desarrollan sitios y aplicaciones web, ya que define cómo se produce esa interacción.

Imaginar Internet sin HTTP sería difícil. No existirían páginas web, URL ni hipervínculos tal como los conocemos. En su lugar, los usuarios tendrían que conocer las direcciones IP exactas de los servidores que alojan los datos deseados y utilizar protocolos de transferencia de datos de nivel más bajo, como TCP/IP. HTTP es el cimiento sobre el cual se construye y funciona la vasta red de Internet.

World Wide Web consiste en la comunicación (ver Fig. 1 ) entre clientes web y servidores web. Los clientes suelen ser navegadores (Chrome, Edge, Safari), pero pueden ser cualquier tipo de programa o dispositivo. Los servidores suelen ser ordenadores en la nube.



Fig. 1 . Solicitud-Respuesta.

### 5.1 Petición y Respuesta HTTP

La interacción entre clientes y servidores se lleva a cabo a través de un proceso de petición y respuesta[4]:

- Un cliente, como un navegador web, envía una petición HTTP al servidor.
- El servidor web recibe la petición y la procesa utilizando una aplicación correspondiente.
- Luego, el servidor devuelve una respuesta HTTP al cliente, que puede contener datos, archivos o información solicitada.
- Finalmente, el cliente, que es el navegador, recibe y procesa la respuesta del servidor.



## 5.2 El ciclo de peticiones HTTP

Un ciclo típico de solicitud y respuesta HTTP sigue este patrón:

1. El navegador solicita una página HTML al servidor, y este responde enviando el archivo HTML solicitado.
2. Posteriormente, el navegador solicita una hoja de estilos CSS para aplicar el diseño a la página, y el servidor responde con el archivo CSS correspondiente.
3. A continuación, el navegador solicita imágenes, como archivos JPG, para mostrar contenido visual en la página, y el servidor envía los archivos de imagen requeridos.
4. Si hay funcionalidades interactivas en la página que requieran código JavaScript, el navegador solicitará archivos JS al servidor, que los proporcionará.
5. Además, si la página necesita datos dinámicos, el navegador solicitará estos datos al servidor, que los enviará en un formato como XML o JSON para ser procesados por el cliente.

Este ciclo describe cómo las solicitudes HTTP y las respuestas se utilizan para cargar y mostrar contenido en un navegador web, permitiendo una interacción fluida entre el cliente y el servidor en la web.

## 5.3 Códigos HTTP

Los códigos HTTP se clasifican en cinco rangos según su significado[7]:

### 1xx: Respuestas informativas

- 100 Continuar: El servidor ha recibido las cabeceras de la solicitud y está listo para recibir el cuerpo de la misma.
- 101 Cambiando protocolos: El servidor acepta el cambio de protocolo propuesto por el cliente.

### 2xx: Éxito

- 200 OK: La solicitud se completó con éxito y el recurso solicitado se encuentra en la respuesta.
- 201 Creado: La solicitud se completó con éxito y se ha creado un nuevo recurso.
- 202 Aceptado: La solicitud ha sido aceptada para procesamiento, pero aún no se ha completado.

- 204 Sin contenido: La solicitud se completó con éxito, pero la respuesta no contiene ningún cuerpo.
- 205 Restablecer contenido: El cliente debe restablecer el contenido de la página.

### **3xx: Redirecciones**

- 300 Múltiples opciones: Existen varias opciones para acceder al recurso solicitado.
- 301 Movido permanentemente: El recurso solicitado ha sido movido permanentemente a otra URL.
- 302 Movido temporalmente: El recurso solicitado ha sido movido temporalmente a otra URL.
- 303 Ver otro: El recurso solicitado se encuentra en otra URL.
- 304 No modificado: El recurso solicitado no ha sido modificado desde la última vez que se solicitó.
- 307 Redirección temporal: El recurso solicitado ha sido movido temporalmente a otra URL y el método de solicitud debe conservarse.

### **4xx: Errores del cliente**

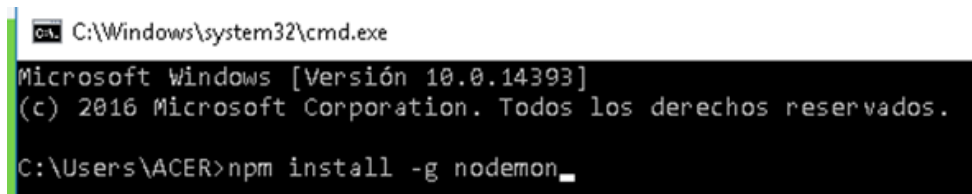
- 400 Solicitud incorrecta: La solicitud del cliente no pudo ser entendida por el servidor debido a una sintaxis incorrecta.
- 401 No autorizado: El cliente no está autorizado para acceder al recurso solicitado.
- 403 Prohibido: El acceso al recurso solicitado está prohibido.
- 404 No encontrado: El recurso solicitado no pudo ser encontrado en el servidor.
- 405 Método no permitido: El método de solicitud utilizado no está permitido para el recurso solicitado.
- 406 No aceptable: El servidor no puede generar una respuesta en el formato solicitado por el cliente.
- 407 Autenticación requerida por proxy: Se requiere autenticación para acceder al recurso solicitado a través de un proxy.
- 413 Entidad demasiado grande: La solicitud del cliente excede el tamaño máximo permitido.
- 414 URL demasiado larga: La URL de la solicitud del cliente es demasiado larga.
- 415 Tipo de medio no admitido: El tipo de medio de la solicitud del cliente no es compatible con el servidor.
- 422 Entidad no procesable: La solicitud del cliente es válida, pero el servidor no puede procesarla.
- 429 Demasiadas solicitudes: El cliente ha enviado demasiadas solicitudes en un período de tiempo corto.
- 451 No disponible: El recurso solicitado no está disponible temporalmente.

**5xx: Errores del servidor**

- 500 Error interno del servidor: El servidor ha encontrado un error inesperado al procesar la solicitud.
- 501 No implementado: El servidor no admite el método de solicitud utilizado.
- 502 Puerta de enlace incorrecta: El servidor ha recibido una respuesta inválida de otro servidor.
- 503 Servicio no disponible: El servidor no está disponible temporalmente.
- 504 Puerta de enlace agotada: El servidor ha agotado el tiempo de espera mientras esperaba una respuesta de otro servidor.
- 505 Versión HTTP no admitida: El servidor no admite la versión HTTP utilizada en la solicitud.

## 6 Desarrollo

Para la elaboración de la práctica se requirió que instaláramos algunas herramientas para desarrollo de páginas web. El primer requerimiento era instalar node.js, tecnología que el equipo ya contaba en sus máquinas. El siguiente requisito consistía en instalar Nodemon, la cual es una herramienta de desarrollo para Node.js que facilita el proceso de desarrollo al reiniciar automáticamente la aplicación cada vez que se detectan cambios en los archivos del proyecto. Esto elimina la necesidad de reiniciar manualmente el servidor después de cada modificación, lo que ahorra tiempo y mejora la productividad del desarrollador. Básicamente monitorea los archivos del proyecto en busca de cambios y reinicia automáticamente la aplicación, lo que permite ver los resultados de las modificaciones de manera rápida y eficiente con solo refrescar la página del navegador. Esta herramienta la instalamos de manera global para no instalarla nuevamente en nuevos proyectos.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.14393]
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ACER>npm install -g nodemon_
  
```

Posteriormente instalamos express en nuestra carpeta de nuestro proyecto inicializando npm. Lo cual nos arrojó los json relacionados a la conexión de express.



```

{} package.json > ...
1  {
2    "name": "2-dpto",
3    "version": "1.0.0",
4    "description": "Hola mundo",
5    "main": "app.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "IPN",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.19.2"
13   }
14 }
15
  
```

```

() package-lockjson > ...
1  {
2    "name": "2-dpto",
3    "version": "1.0.0",
4    "lockfileVersion": 3,
5    "requires": true,
6    "packages": {
7      "": {
8        "name": "2-dpto",
9        "version": "1.0.0",
10       "license": "ISC",
11       "dependencies": {
12         "express": "^4.19.2"
13       }
14     },
15     "node_modules/accepts": {
16       "version": "1.3.8",
17       "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.8.tgz",
18       "integrity": "sha512-PyAthTa2m2VKXu vSD3DPC/Gy+U+sOA1LAuT8mkRu vw+NACSaeXEQ+NHcVF7rON16qc axV3Uue
19       "dependencies": {
20         "mime-types": "~2.1.34",
21         "negotiator": "0.6.3"
22       },
23       "engines": {
24         "node": ">= 0.6"
25       }
26     }
27   }

```

En nuestro script de JavaScript configuramos nuestro programa para utilizar express, indicar que puerto usaremos y el número del hostname. En la parte final nos aseguramos que la conexión es estable.

```

JS app.js > app.post('/otraRuta') callback
1  var express = require('express');
2  var app = express();
3  const port = 8000;
4  const hostname = '127.0.0.1';
5

```

```

31
32  app.listen(port, hostname, () => {
33    console.log(`Servidor corriendo en http://${hostname}:${port}`);
34  });
35

```

Posteriormente configuramos dos rutas raíces y dos rutas diferentes agregando los métodos GET y POST. Elementalmente la ruta raíz se refiere a la ruta principal de nuestro URL en cuanto a raíces configuradas, poseemos un sin fin de formas de nombrar otra sección de nuestra página.

Los métodos GET nos sirven para crear una comunicación entre el servidor y el back end. En los ejemplos de ejecución sencillamente regresamos texto plano.

```
5
6  app.get('/', (req, res) => {
7    res.status(200)
8    res.send('Hola usuario humano que tal? Probando ');
9  });
10
```

```
10
11  app.get('/', (req, res) => {
12    res.status(200)
13    res.send('Que te trae por aqui?')
14  });
15
```

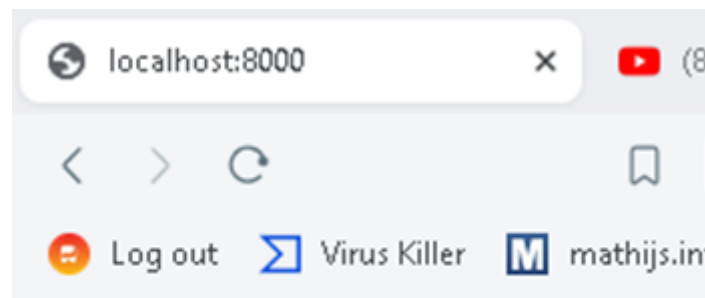
Cabe añadir que en todas las funciones que aplican un método enviamos también un código de estado, en nuestro caso el 200, el cual expresa que la operación es exitosa.

Finalmente para los dos métodos post que creamos uno devuelven un documento tipo json con valores aleatorios y el otro un texto plano. Para poder ejecutar estos métodos es necesario usar la aplicación de postman, dado que el método post es un método más seguro. Y no aparece en el encabezado de la página. Este tipo de método es muy utilizado cuando accedes a elementos de Front End así como datos obtenidos de algún formulario.

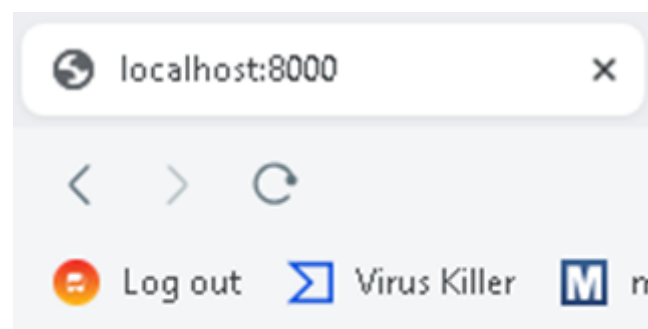
```
15
16  app.post('/otraRuta', (req, res) => {
17    res.status(200)
18    res.send('Solicitud de POST recibida');
19  });
20
```

```
20
21 app.post('/jsonAleatorio', (req, res) => {
22   res.status(200)
23   const randomObject = {
24     numero: Math.floor(Math.random() * 100),
25     cadena: Math.random().toString(36).substring(7),
26     booleano: Math.random() < 0.5
27   };
28   res.json(randomObject);
29 });
30
31
```

Como podemos ver en las capturas de pantallas de abajo nuestro navegador nos muestra el mensaje que nosotros declaramos en los métodos GET.

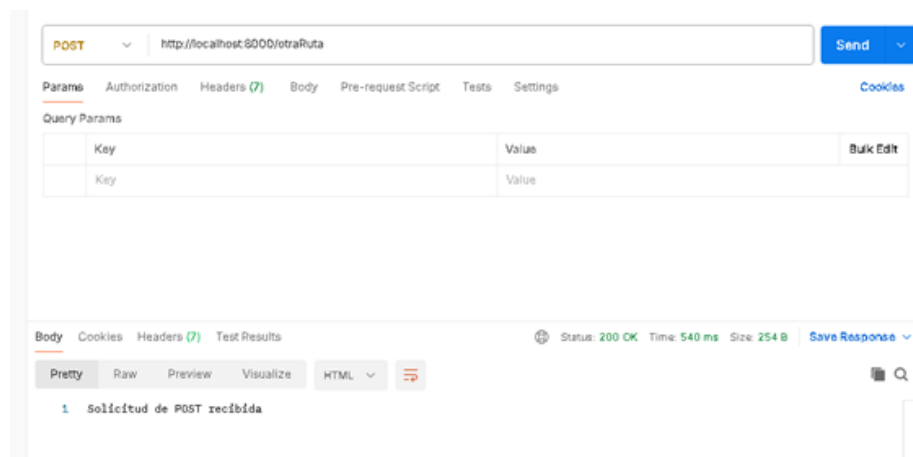
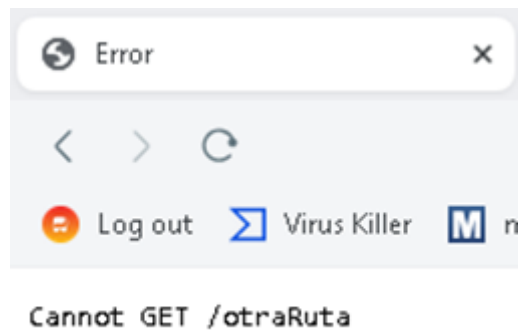


Hola usuario humano que tal? Probando

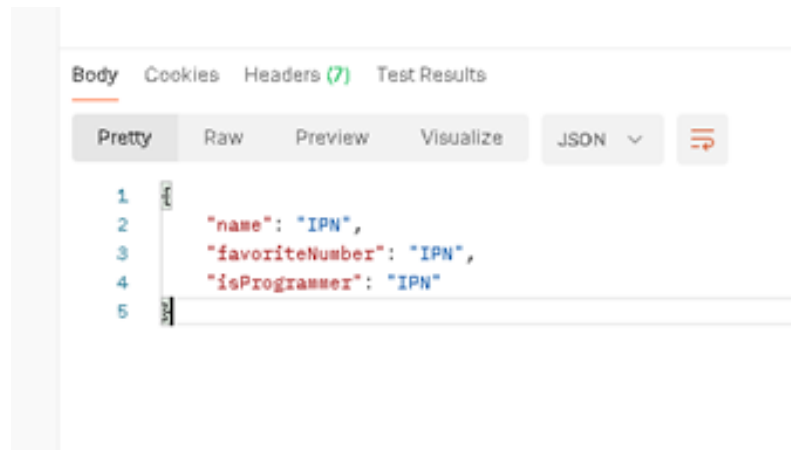


Que te trae por aqui?

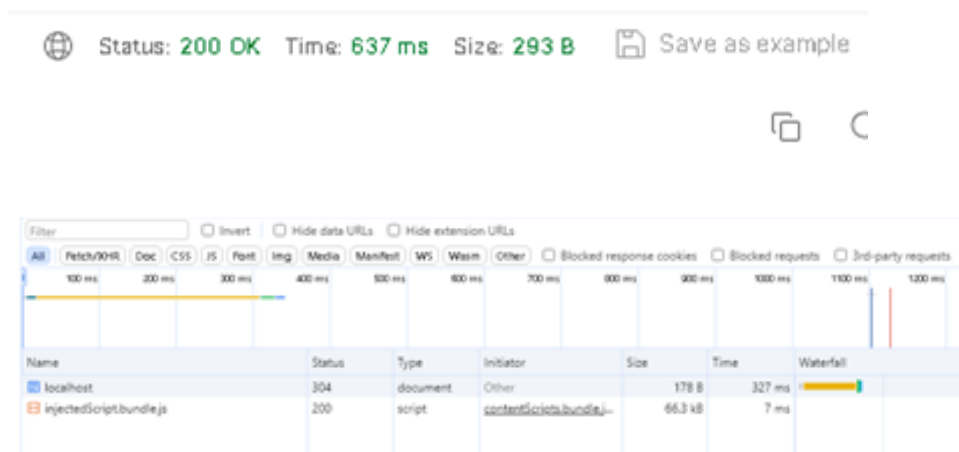
Para ejecutar y visualizar los resultados del método post, es necesario instalar postman, ya que como mencionamos antes este método nos permite tener un poco más de privacidad y no visualizar el resultado sino en el body.







Igualmente en nuestros métodos como GET y POST enviamos el código de estado 200. El cuál lo vemos en postman o en el navegador.



## 7 Conclusiones

Para nosotros los desarrolladores de aplicaciones y software enfocados en inteligencia artificial, será de suma importancia aprender a desarrollar o emplear nuestras aplicaciones en un entorno web, por ende, el uso y manipulación de los servidores además de su gran variedad de extensiones, herramientas, frameworks, etc es algo indispensable. Si bien en un comienzo (teniendo en cuenta que no se ha trabajado con frameworks) JavaScript y sus frameworks pueden tener una considerable dificultad en los comienzos, esta va siendo más fácil de comprender e implementar conforme más código se escriba (como en todo desarrollo), además de que Javascript tiene una inmensa comunidad de programadores, y por lo tanto guías, tutoriales y documentación. En el caso de esta práctica la parte más confusa fue el caso de Postman, si bien comprender los códigos no es difícil, el cómo devolverlos, las circunstancias o el cuando puede ser la parte desafiante.

## Referencias

1. *Servidores web en comparación con servidores de aplicaciones: diferencia entre servidores tecnológicos - AWS.* (n.d.). Amazon Web Services, Inc. <https://aws.amazon.com/es/compare/the-difference-between-web-server-and-application-server/>
2. *Node.js-Introduction.* (n.d.). [https://www.tutorialspoint.com/nodejs/nodejs\\_introduction.htm](https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm)
3. Huawei, <https://forum.huawei.com/enterprise/es/%C2%BFqu%C3%A9-es-un-servidor-web/thread/674996634846642176-667212884846981120>, last accessed 23/04/24.
4. *What is HTTP.* (n.d.). [https://www.w3schools.com/whatis/whatis\\_http.asp](https://www.w3schools.com/whatis/whatis_http.asp)
5. Manuel, J. (2024, January 21). *¿Qué son los servidores web y por qué son necesarios?* Duplika. <https://duplika.com/blog/que-son-los-servidores-web-y-por-que-son-necesarios/>
6. *Qué es Express.JS y primeros pasos.* (2022, March 3). IfgeekthenNTTdata. <https://ifgeekthen.nttdata.com/es/que-es-expressjs-y-primeros-pasos>.
7. *Códigos de estado de respuesta HTTP - HTTP | MDN.* (2022, November 26). MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTTP/Status>

## Anexo.

```
var express = require('express');
var app = express();
const port = 8000;
const hostname = '127.0.0.1';

app.get('/', (req, res) => {
```

```

        res.status(200).send('Hola usuario humano que tal?
Probando ');
    });

app.get('/otroGet', (req, res) => {
    res.status(200).send('Que te trae por aqui?')
});

app.post('/otraRuta', (req, res) => {
    res.status(200)
    res.send({
        "name" : "IPN",
        "favoriteNumber" : "IPN",
        "isProgrammer" : "IPN"
    });
});

app.post('/jsonAleatorio', (req, res) => {
    res.status(200)
    const randomObject = {
        numero: Math.floor(Math.random() * 100),
        cadena:
Math.random().toString(36).substring(7),
        booleano: Math.random() < 0.5
    };
    res.json(randomObject);
});

app.listen(port, hostname, () => {
    console.log(`Servidor corriendo en
http://${hostname}:${port}`);
});

```

