



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*

Clase 3: Servidores web - el módulo HTTP - Manejo de errores

Agenda de hoy

- A. Servidores web
 - a. concepto
 - b. peticiones HTTP
 - c. Estados
- B. El módulo HTTP
 - a. Cómo crear un servidor web en Node.js
 - b. Particularidades del módulo HTTP
- C. Rutas
 - a. qué son las rutas
 - b. cómo escuchamos rutas
 - c. limitaciones
- D. Errores
 - a. manejo de errores de petición
 - b. manejo de errores en las rutas



Servidores web

Finalizamos el repaso general de los conceptos más importantes del lenguaje JS, como para estar a tono en lo que respecta a la creación de proyectos backend.

Vamos entonces a adentrarnos en el uso intensivo de Node.js y JavaScript, para sacar el máximo provecho de sus características apropiadas para el universo de servidores.

Pero primero, repasemos el concepto de servidores web.



Servidores web

Servidores web

Los servidores web son programas o sistemas informáticos **que proveen contenido y servicios web**, usualmente, a través de Internet.

Un servidor web **recibe y procesa las solicitudes** de los usuarios, y **luego responde con los datos solicitados**, como páginas web, imágenes, videos y otros archivos.



Servidores web

También, **alojan y sirven archivos y datos a través de un protocolo de transferencia de hipertexto (HTTP)** o su versión segura (HTTPS).

Los sitios web se crean y se cargan en un servidor web, y los usuarios acceden a ellos mediante un navegador web, que solicita estos archivos al servidor, para mostrarlos en el browser.



Servidores web

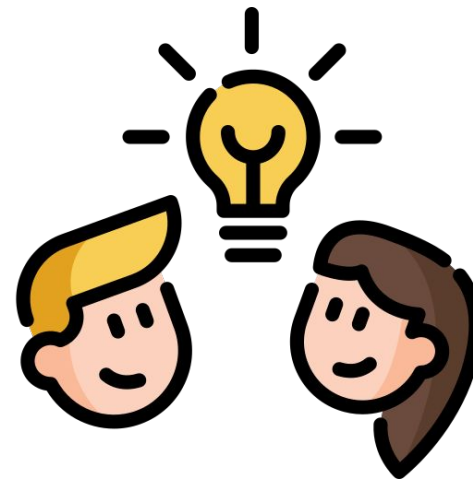
En una aplicación de backend, el servidor web es responsable de recibir y responder a las solicitudes de los clientes, y de coordinar la interacción entre los diferentes componentes de la aplicación.

Estas solicitudes son recibidas a través de HTTP y las enruta a la aplicación adecuada para su procesamiento.



Servidores web

En resumen, el servidor web es una parte esencial de las aplicaciones de backend, ya que proporciona la infraestructura necesaria para que la aplicación se comuniquen con los clientes y gestione la complejidad de la interacción de la aplicación.

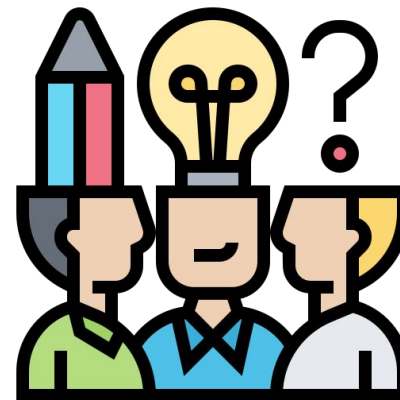


Servidores web

¿Y en Node.js, qué papel cumple un servidor web? 🤔

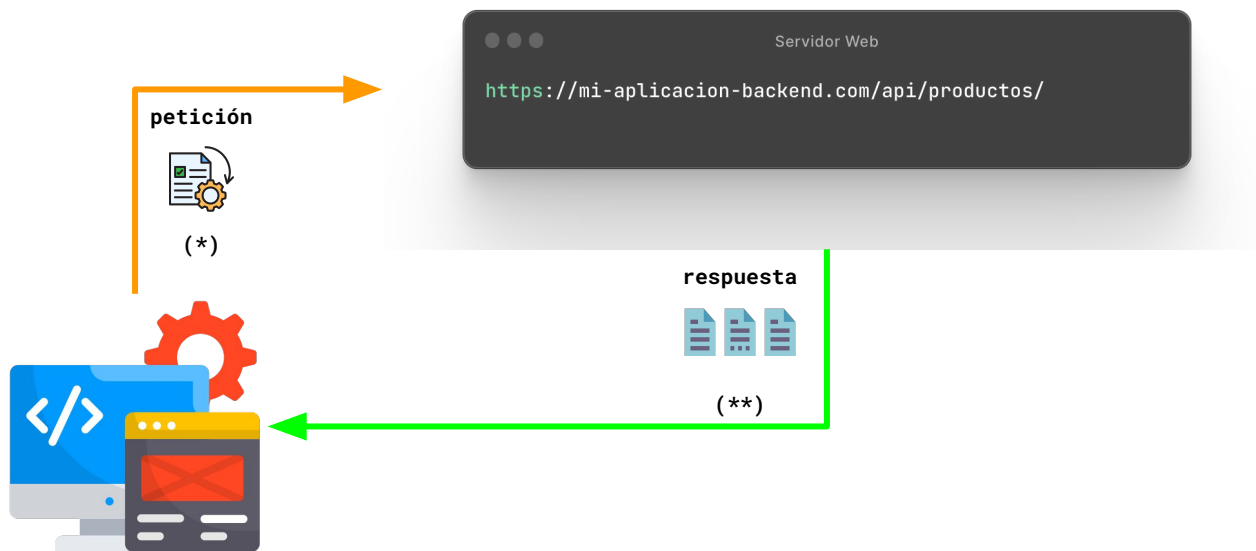
En nuestro caso, no necesitamos un servidor web como los mencionados anteriormente. Contamos con las herramientas necesarias para crear uno, de acuerdo a nuestra necesidad.

Para lograrlo, **combinaremos a Node.js con el lenguaje JavaScript y así crear nuestro propio Webserver.**



Peticiones HTTP

Peticiones HTTP



Peticiones HTTP

Las peticiones HTTP (*Protocolo de Transferencia de Hipertexto*) son una **forma común de comunicación** entre aplicaciones cliente y servidor.

En el contexto de las aplicaciones backend, las peticiones HTTP se utilizan para solicitar datos o realizar acciones en el servidor que aloja la aplicación.



Peticiones HTTP

Se dividen en varias partes, incluyendo:

Método: El método HTTP utilizado para la solicitud, como GET, POST, PUT, DELETE, etc.

URL: La URL (*Uniform Resource Locator*) identifica el recurso al que se está accediendo en el servidor.

Encabezados: Los encabezados HTTP se utilizan para proporcionar información adicional sobre la solicitud, (*tipo de contenido que se espera*).

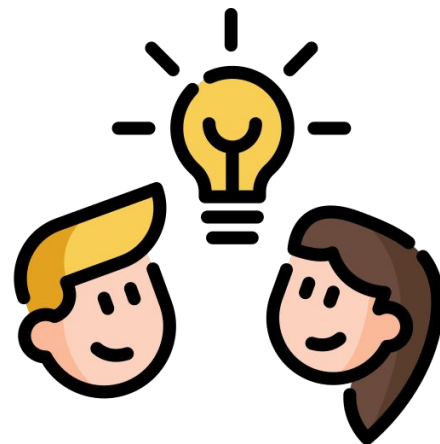
Cuerpo: El cuerpo de la solicitud puede incluir datos que se envían al servidor, como parámetros de consulta o datos de formulario.



Peticiones HTTP

En el contexto de las aplicaciones backend, **las peticiones HTTP se utilizan para acceder a recursos alojados en el servidor**, como bases de datos, archivos, servicios web y otros recursos.

Los servidores backend procesan estas **solicitudes y devuelven respuestas** en función de los datos solicitados y las acciones realizadas.

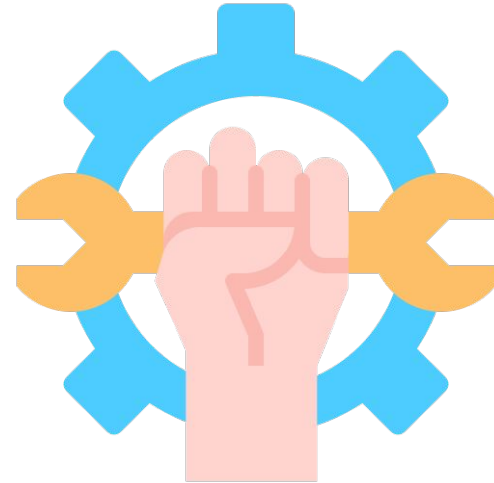


Estados

Estados

Cuando se realiza una solicitud HTTP, el servidor responde con un código de estado HTTP para indicar el resultado de la solicitud.

Los códigos de estado HTTP **son números de tres dígitos** que se utilizan para indicar si la **solicitud se ha completado** correctamente o si ha habido algún problema.



Estados

Hay varios códigos de estado HTTP, pero los más comunes son:

ESTADO	DESCRIPCIÓN
200 OK	Indica que la solicitud se ha procesado correctamente y se devuelve la respuesta solicitada.
201 Created	Indica que la solicitud ha sido procesada correctamente y que se ha creado un nuevo recurso en el servidor.
400 Bad Request	Indica que la solicitud no se pudo entender o procesar debido a un error en la sintaxis de la solicitud.
401 Unauthorized	Indica que el usuario no tiene autorización para acceder al recurso solicitado.
403 Forbidden	Indica que el servidor rechaza la solicitud porque el usuario no tiene permisos para acceder al recurso.
404 Not Found	Indica que el servidor no pudo encontrar el recurso solicitado.
500 Internal Server Error	Indica que se produjo un error en el servidor mientras procesaba la solicitud.
503 Service Unavailable	Indica que el servidor no está disponible en este momento para procesar la solicitud debido a una sobrecarga o mantenimiento.



Estados

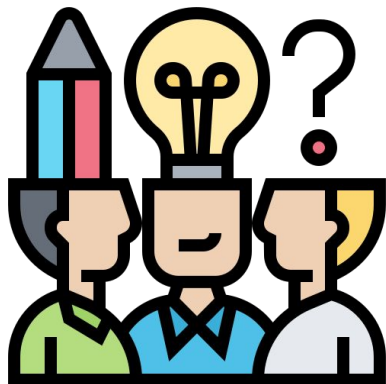
Es importante tener en cuenta que los códigos de estado HTTP no son exclusivos de las aplicaciones backend y se aplican a cualquier solicitud HTTP realizada a cualquier servidor.

Sin embargo, en el contexto de las aplicaciones backend, estos códigos se utilizan para indicar el resultado de las solicitudes realizadas a recursos alojados en el servidor.



Iniciar un Proyecto Node.js

Ejercicio práctico



Abrimos un espacio de trabajo, donde desarrollaremos los siguientes puntos:

- ✓ Crear una carpeta para tu proyecto
- ✓ Inicializar el proyecto con npm
- ✓ Verificar/Configurar el archivo package.json
- ✓ Crear un archivo de entrada

Lo trabajaremos como una receta, para no olvidar ninguno de los ingredientes.

La profe hará un resumen de los comandos y estructura de archivos base del proyecto Node.js.

El módulo HTTP

El módulo HTTP

Este módulo es una parte integral del núcleo de Node.js y proporciona un conjunto de funcionalidades en todo lo que respecta, no solo a crear, sino también gestionar servidores web y aplicaciones cliente HTTP.

A continuación, veremos algunas de las propiedades y métodos más importantes del módulo:



El módulo HTTP

Propiedades y Métodos	Descripción
http.createServer()	Este método crea un nuevo servidor HTTP. Toma una función de devolución de llamada como argumento que se ejecuta cada vez que el servidor recibe una solicitud.
server.listen()	Este método permite al servidor escuchar las solicitudes en el puerto especificado.
server.on()	Este método permite establecer un controlador de eventos para el servidor. Por ejemplo, se puede utilizar para manejar eventos como <i>"request"</i> (cuando se recibe una solicitud) y <i>"connection"</i> (cuando se establece una conexión).
request.method	Esta propiedad indica el método HTTP utilizado en la solicitud (por ejemplo, <i>"GET"</i> o <i>"POST"</i>).
request.url	Esta propiedad indica la URL solicitada.
response.writeHead()	Este método permite escribir el encabezado de la respuesta HTTP.
response.end()	Este método permite finalizar y enviar la respuesta al cliente.

Estas propiedades y métodos, entre muchos más, son los que conforman el módulo HTTP. Trabajaremos inicialmente con algunos de ellos, para entenderlos bien.

El módulo HTTP

En la web de Node.js encontraremos la documentación oficial que nos proveerá la información completa de todas las propiedades y métodos que disponemos en este módulo.

No olvides consultarla ante cualquier duda que te surja: <https://nodejs.org/api/http.html>



El módulo HTTP

Con el entorno listo para comenzar a programar y el principal actor que entrará en escena, comencemos a desarrollar el código base de nuestro primer servidor web.

Trataremos este proceso como una receta, lo cual será la forma más fácil de organizar nuestras tareas.



El módulo HTTP

01

Sumar a nuestro proyecto Node, el módulo HTTP necesario para la aplicación

02

Definiremos un puerto `{PORT}` donde el servidor “escuchará” las peticiones

03

Crearemos el servidor web, y lo pondremos a escuchar peticiones

04

Ante una petición, devolveremos una respuesta simple basada en texto plano



Crear el servidor web

Crear el servidor web

El método **require()** se ocupa de sumar el módulo HTTP a nuestro proyecto. Cumple una función similar a cuando instancia mos una clase u objeto.

El puerto que utilizaremos es el **3000**.

El método **createServer()**, crea el servidor.

Por último, el método **listen()** comienza a escuchar peticiones en el puerto que le definimos.

```
Server Web

const http = require('http');
const PORT = 3000

const server = http.createServer((request, response) => {
})

server.listen(PORT, () => {
  console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
})
```

Crear el servidor web



```
const http = require('http');  
const PORT = 3000
```

El **puerto numérico** que definimos corresponde a un puerto específico de la computadora, habitualmente administrado por el sistema operativo, el cual se define para que una tarea o proceso escuche peticiones de otras aplicaciones (*llamadas clientes*).

Crear el servidor web

El método **createServer()** recibe una función como parámetro y, a su vez, ésta recibe dos objetos como parámetros: request y response.

El primero de estos objetos (**request**), es el cual canaliza (*recibe*) las peticiones en cuestión de cada cliente (*navegadores web, apps, etc.*).

response, por su parte, es quien se ocupará de brindar las respuestas a el o los clientes que le realizan peticiones.

```
Server Web  
  
const server = http.createServer((request, response) => {  
  
  })
```

Crear el servidor web

```
Server Web

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end('Hola, mundo!');
})
```

Nos queda por delante responder la petición al cliente que la realizó. Junto a la respuesta **debemos informarle**, el **código de estado**, **qué contenido le estamos enviando** y, finalmente, **la respuesta** que entregamos a su petición.

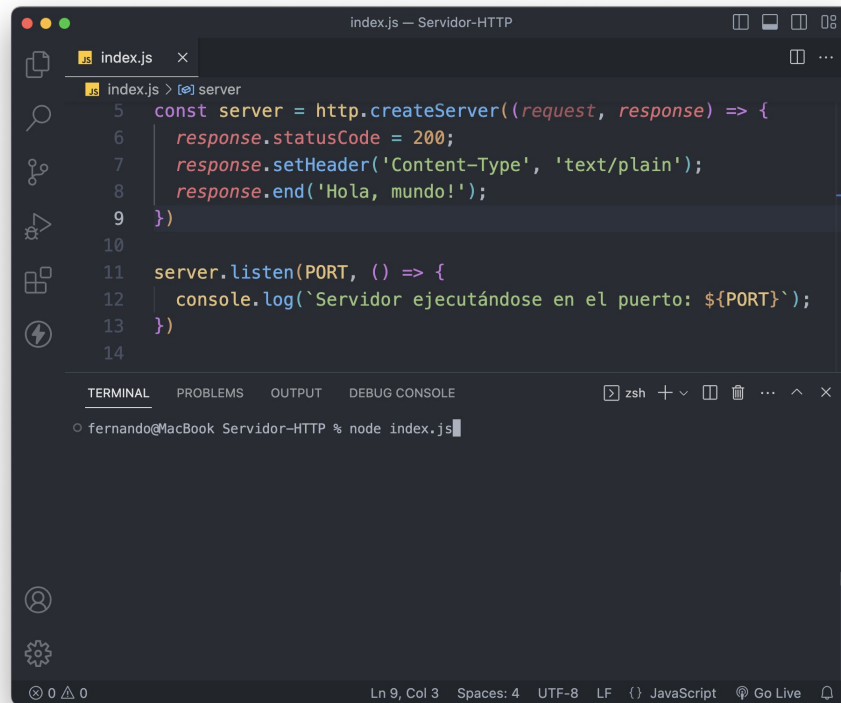
Todo esto, de la mano del objeto **response**.

Ignición

Ignición

Para poner en marcha el servidor web, debemos utilizar la ventana Terminal y el comando **node**.

Como siempre, el parámetro que debe recibir es el nombre del archivo JavaScript donde está el código que le da vida al webserver.

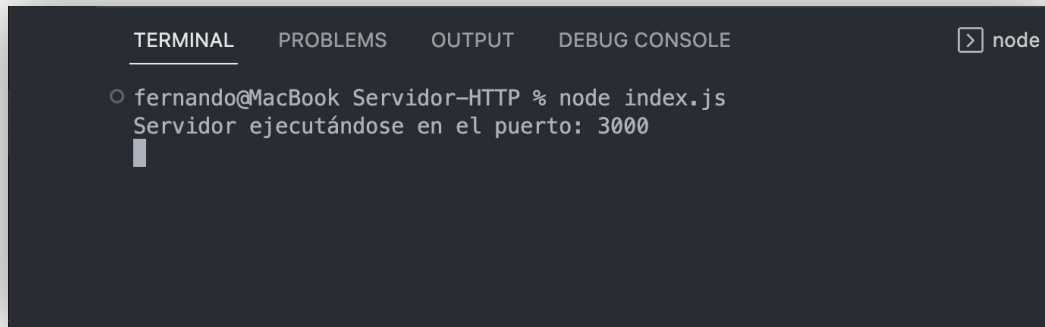


```
index.js — Servidor-HTTP
index.js x
index.js > server
5  const server = http.createServer((request, response) => {
6    response.statusCode = 200;
7    response.setHeader('Content-Type', 'text/plain');
8    response.end('Hola, mundo!');
9  })
10
11  server.listen(PORT, () => {
12    console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
13  })
14

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
fernando@MacBook Servidor-HTTP % node index.js
```



Ignición



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  > node
fernando@MacBook Servidor-HTTP % node index.js
Servidor ejecutándose en el puerto: 3000
```

Al ejecutar este comando en la ventana Terminal veremos el mensaje de log de nuestra aplicación, el cual nos informa en qué puerto se ejecuta el servidor.

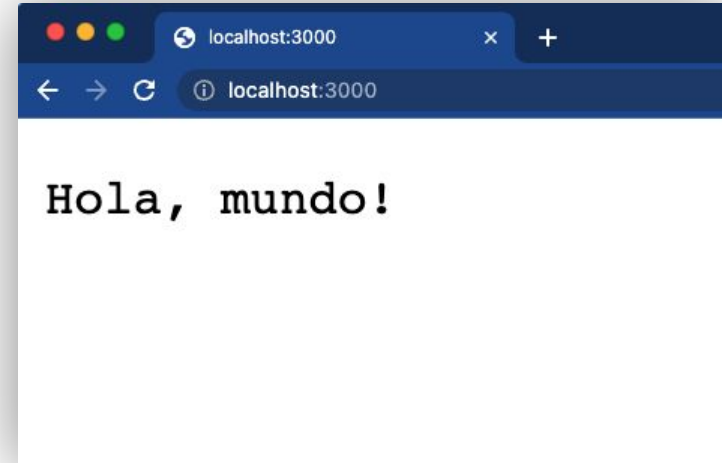
Nos vamos al navegador web para ejecutar la “Prueba de fuego”.



Ignición

Abrimos un navegador web y escribimos en una nueva pestaña, la URL correspondiente a nuestro servidor web, y pulsamos Enter.

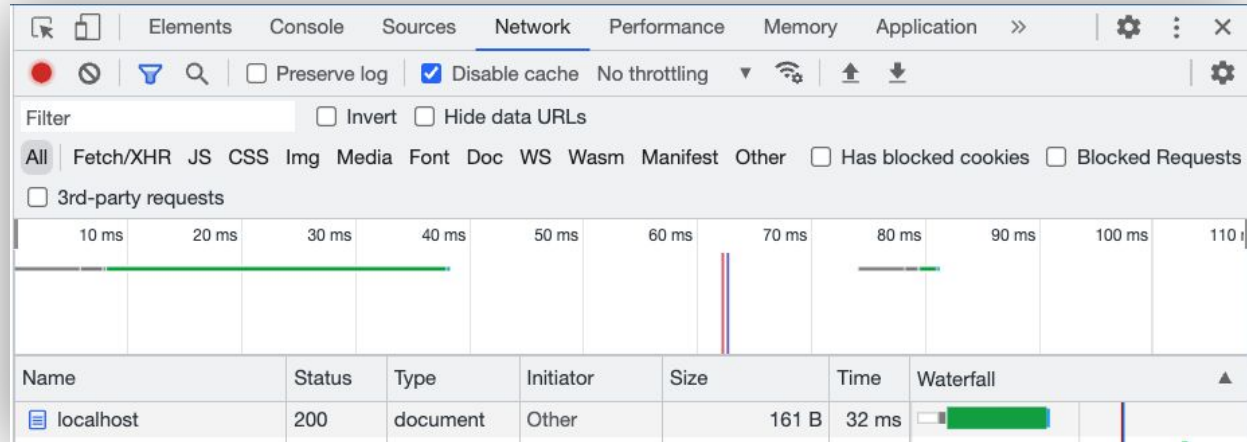
Si todo va bien, debemos visualizar un texto simple, como el que agregamos en nuestra respuesta de servidor.



Ignición

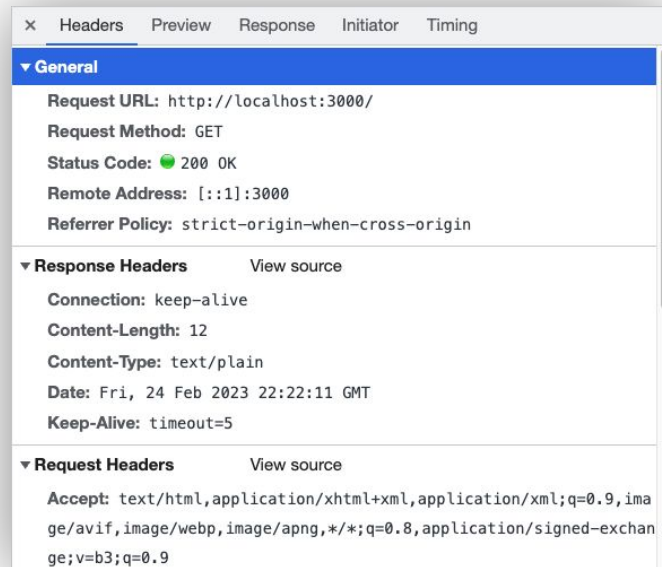
Abrimos **DevTools** en el navegador web y, en la **pestaña Network**, encontramos información relacionada a la respuesta recibida por parte del servidor:

Tiempo de respuesta, Status, Tipo de archivo de respuesta, entre otros datos adicionales.



Ignición

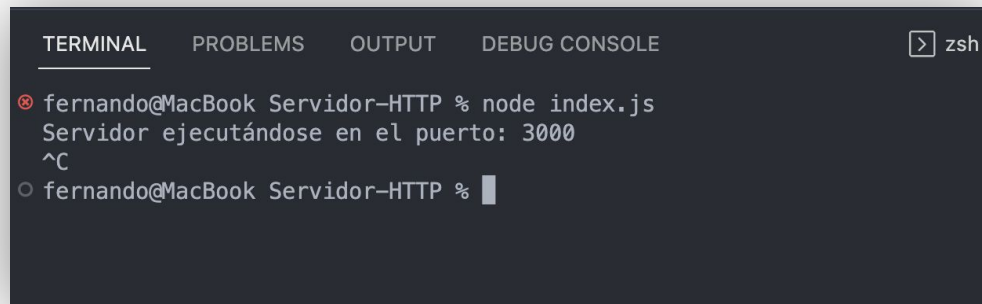
Seleccionando el archivo de respuesta del servidor, accedemos a un detalle mucho más amplio de “*todo el diálogo*” que ocurre entre el webserver y el cliente, en este caso, el navegador web en cuestión.



Ignición

Para detener el webserver, **debemos darle foco a la ventana Terminal y presionar la combinación de teclas CTRL + C.**

Así interrumpimos la ejecución del mismo.



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  > zsh
fernando@MacBook Servidor-HTTP % node index.js
Servidor ejecutándose en el puerto: 3000
^C
fernando@MacBook Servidor-HTTP %
```

Ignición

El código completo del servidor web.

```

Servidor Web

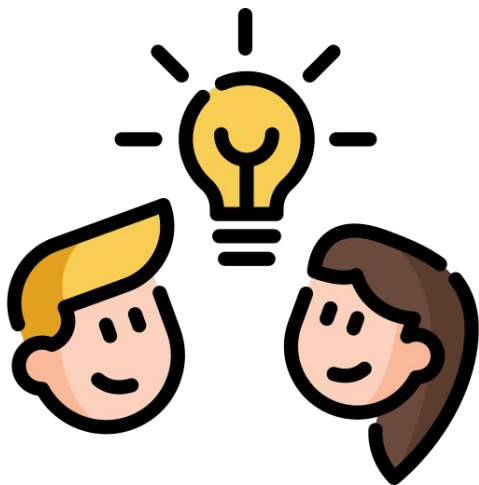
const http = require('http');

const PORT = 3000

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end('Hola, mundo!');
})

server.listen(PORT, () => {
  console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
})
```


Herramientas adicionales



Existen otras tantas herramientas que nos permiten trabajar de forma más efectiva entre el “*ida y vuelta*” del servidor web y la aplicación cliente, y viceversa.

Las iremos incorporando más adelante, de manera gradual, a medida que sigamos aprendiendo los diferentes actores que aún deben entrar en escena.

Rutas

Rutas

En el contexto de las aplicaciones de backend con Node.js, **una ruta**, refiere a la URL utilizada para acceder a un recurso específico alojado en el servidor. Éstas se definen en el servidor web para que el mismo pueda recibir solicitudes HTTP y dirigir las al controlador adecuado para su procesamiento.



Por ejemplo, si se está construyendo una aplicación de blog, es posible que deseemos tener una ruta **/posts** que devuelva una lista de publicaciones de blog, y una ruta **/posts/:id** que devuelva una publicación de blog específica en función de su **ID**.

Rutas

Basándonos en el servidor web que desarrollamos hasta aquí, podemos imaginar un ejemplo de diferentes rutas a partir de estas URL modelo.

El módulo HTTP permite controlar desde el servidor web creado, peticiones a diferentes rutas, y devolver una respuesta personalizada para cada una de ellas.

```
https://localhost:3000/           //raíz del sitio
https://localhost:3000/nosotras   //ruta "nosotras"
https://localhost:3000/cursos     //ruta "cursos"
https://localhost:3000/contacto   //ruta "contacto"
```

Rutas

Los parámetros que recibe el método **createServer()** dentro de la función son siempre una petición (**request**), y el otro parámetro la respuesta (**response**).

```
Server Web  
  
const server = http.createServer((request, response) => {  
  
})
```

Estos objetos en cuestión, suelen ser abreviados como **req** y **res**. Podemos sumarnos a esta forma de definirlos, porque es la que vamos a encontrar habitualmente en cualquier ejemplo que busquemos por Internet.

```
Server Web  
  
const server = http.createServer((req, res) => {  
  
})
```

Rutas

Del parámetro **request** (*req*), usado al invocar el método **createServer**, podemos rescatar información adicional enviada originalmente por el cliente.

En nuestro primer servidor web, sin importar la ruta que se peticione desde el cliente (*navegador web*), la respuesta del servidor siempre iba a ser la misma.

Esto se da porque no leemos y analizamos información sobre la ruta peticionada.



```
const server = http.createServer((req, res) => {  
  })
```



Rutas

```
if (req.url === '/') {  
  console.log("Respuesta específica...");  
}
```

A través de **la propiedad url** la cual forma parte del objeto **request**, **podemos obtener la ruta peticionada desde un cliente**, y con esto, enviarle información única a cada ruta.

Para realizar esta tarea, una estructura de **if - else** encadenados, es el aliado ideal.

Rutas

Armemos la siguiente estructura para definir qué respuesta le daremos a cada una de las rutas aquí representadas.

Recuerda utilizar los métodos **.writeHead()** y **.end()** en cada respuesta, y utilizar el **estado de respuesta 200** para cada ruta, excepto para las rutas inexistentes, donde el estado de respuesta será **404**.

```
const server = http.createServer((req, res) => {  
  if (req.url === '/') {  
    //respuesta para la ruta raíz  
  } else if (req.url === '/nosotras') {  
    //respuesta para la ruta /nosotras  
  } else if (req.url === '/cursos') {  
    //respuesta para la ruta /cursos  
  } else if (req.url === '/contacto') {  
    //respuesta para la ruta /contacto  
  } else {  
    //respuesta para rutas inexistentes  
  }  
});
```


Errores en rutas

Ten en cuenta que la ruta para la URL raíz debe ser la primera condición en el bloque de declaraciones if ... else if, para que sea verificada primero, ya que todas las demás rutas comienzan con una barra diagonal (/).

Si la ruta para la URL raíz se coloca después de las otras rutas, nunca se ejecutará.



Errores en rutas

Y para manejar cualquier error relacionado a un recurso o ruta inexistente, debemos cerrar el encadenamiento de if - else if, con la sentencia 'else'.

Todo recurso que no esté definido dentro de esta estructura, será procesado por 'else', e interpretado como un error del tipo **404** - 'recurso no encontrado'.



Desafío

Crear un servidor web utilizando el módulo HTTP.

El mismo deberá contener una serie de rutas específicas, y responder con HTML acorde, cada vez que se peticiona a cada ruta.

También deberás controlar las rutas erróneas, respondiendo con un texto acorde ante una petición errónea.



Prácticas

Definiremos una constante **PORT**, con el valor **3008**. En el servidor web, debemos tener definidas a las siguientes rutas:

- “/”
- “/cursos”
- “/contacto”

Sestea el método **header()** en c/u de ellas, para enviar **content-type** en formato **HTML**. En el método **end()** debemos enviar un tag **H1** con un título descriptivo para la ruta que estamos navegando. Ejemplo:

- “*Bienvenidas a nuestra web*” => para la raíz “/” del sitio
- “*Bienvenidas a nuestra sección cursos*” => para /cursos

Recordemos agregar el control de rutas inexistentes, respondiendo con un mensaje acorde, pero en formato **TEXTO PLANO**.



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*