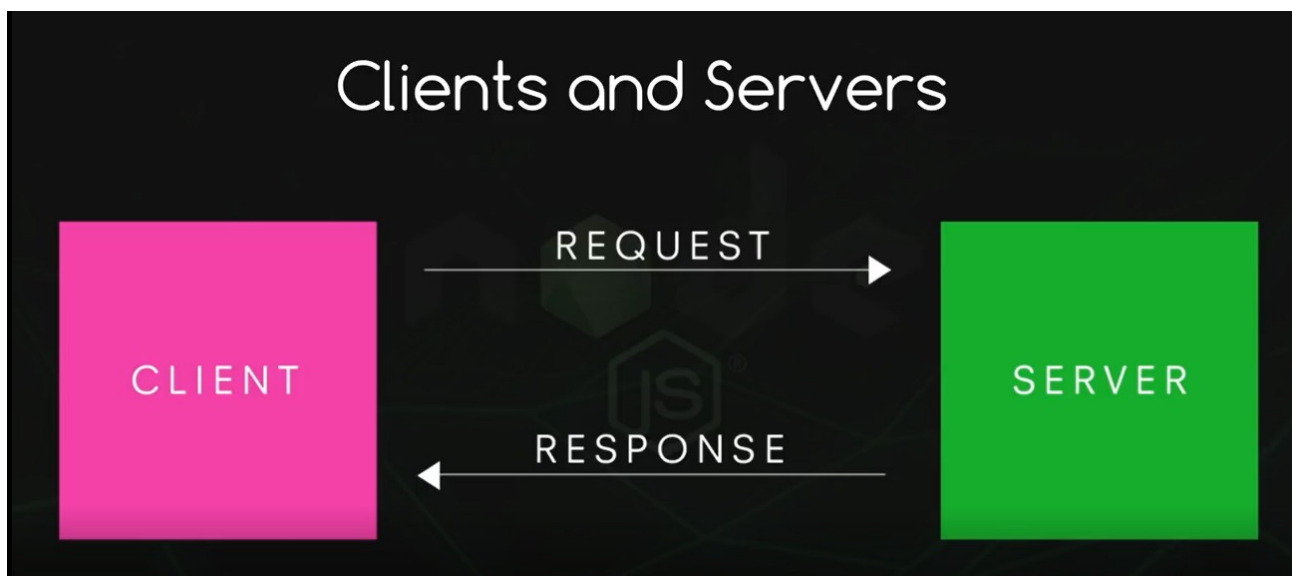


Índice

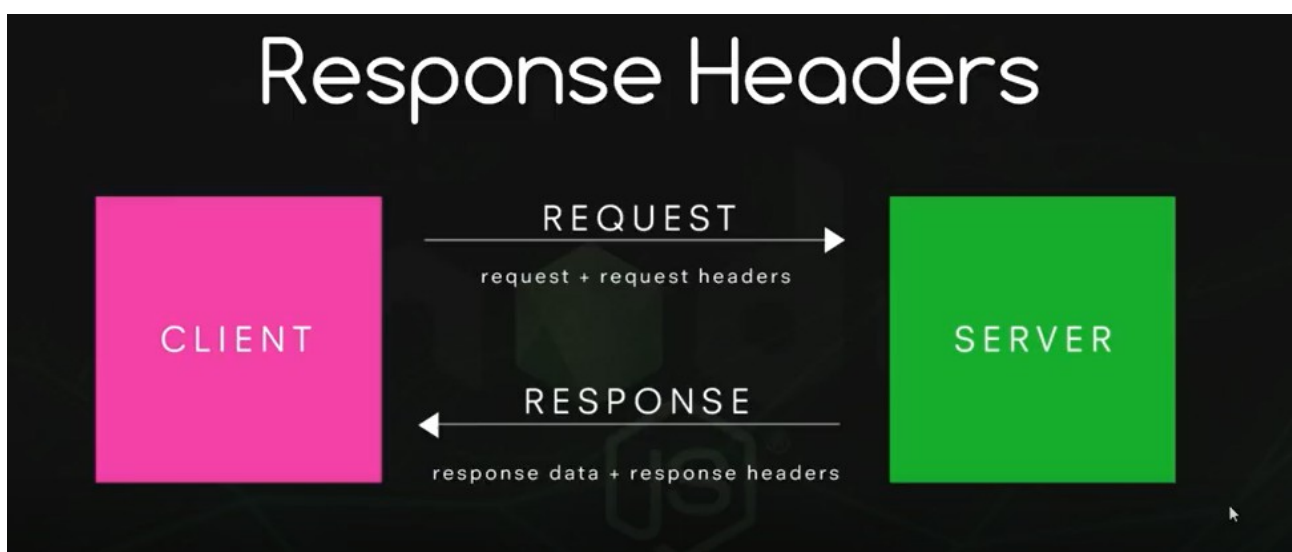
1. Servidores
2. Sirviendo JSON

Servidores

En primer lugar, recordemos que un cliente que realiza una petición a un servidor, que a su vez envía una respuesta, se comunica con el mismo a través de protocolos: reglas predefinidas para establecer comunicaciones a través de sockets, que son una especie de canal de comunicación mediante TCP o puertos UDP. Cada protocolo de capa de aplicación TCP/IP realiza una función: transmisión de información web, transferencia de archivos, correo electrónico, etc.



La información adicional se envía a través de encabezados.



Por ejemplo, en la respuesta, el servidor le dice al cliente el tipo de contenido que envía para saber qué hacer con él, un navegador mostrará la información html como una página web pero no se mostrará un archivo de

texto o json. El estado del envío también aparece en los encabezados de respuesta: 200 si todo correcto, error 404, etc.

En Node podemos crear servidores http usando el módulo http.

```
7servidores.js
1  const http=require('http');
2
3
```

Para crearlo contamos con el método `createServer` de este módulo, que toma como parámetro una función a su vez con dos parámetros, uno que almacena la petición realizada y otro la respuesta.

```
var server=http.createServer(function (req,res) {
});
```

http.createServer([options][, requestlistener])

[\[src\]](#)

► History

- `options` <Object>
 - `IncomingMessage` <`http.IncomingMessage`> Specifies the `IncomingMessage` class to be used. Useful for extending the original `IncomingMessage`. **Default:** `IncomingMessage`.
 - `ServerResponse` <`http.ServerResponse`> Specifies the `ServerResponse` class to be used. Useful for extending the original `ServerResponse`. **Default:** `ServerResponse`.
- `requestListener` <Function>
- Returns: <`http.Server`>

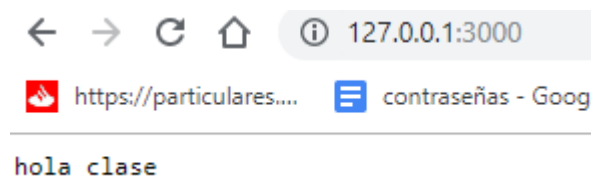
Returns a new instance of `http.Server`.

The `requestListener` is a function which is automatically added to the `'request'` event.

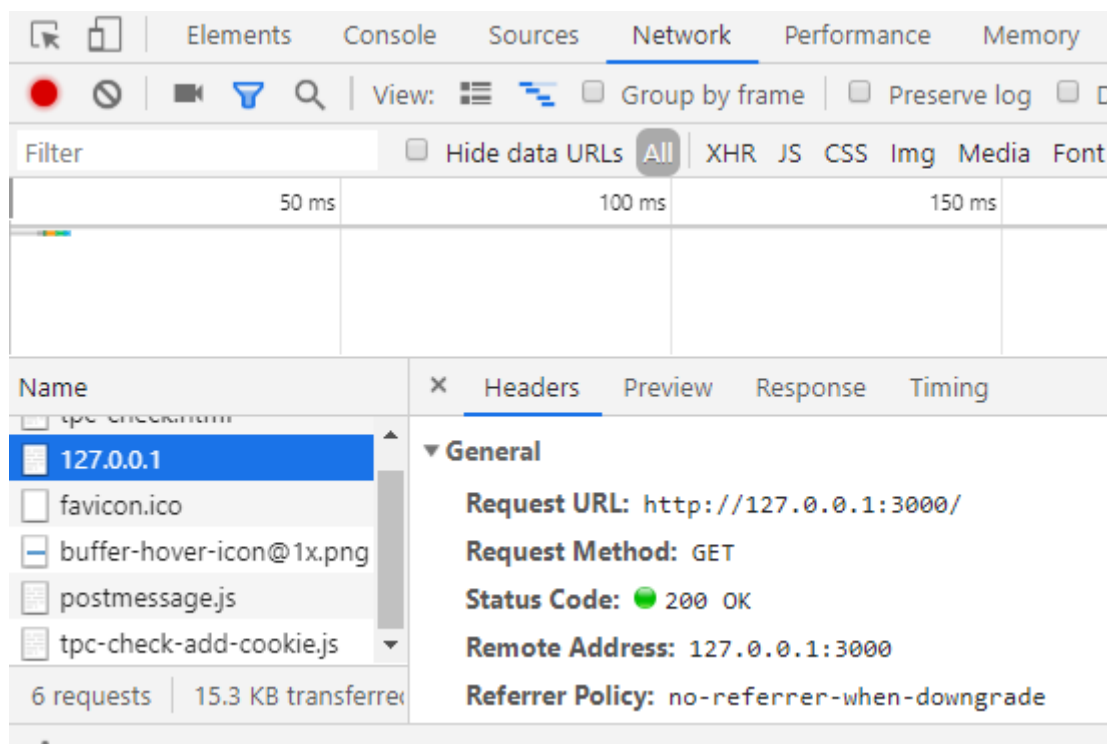
Vamos a especificar el encabezado de respuesta donde diremos que se envía texto sin formato y crearemos el texto sin formato en sí. También es necesario definir un puerto y la IP, usaremos 3000 y como servidor que creamos ahora es local 127.0.0.1.

```
var server=http.createServer(function (req,res) {  
  res.writeHead(200, {'Content-type':'text/plain'});  
  res.end('hola clase');  
});  
  
server.listen(3000,'127.0.0.1');
```

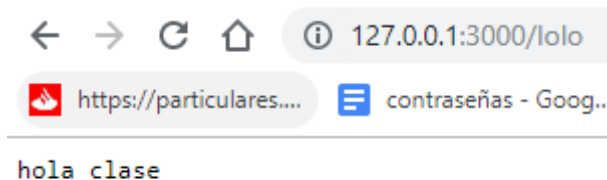
Ahora vamos a un navegador web y escribimos



y si hacemos clic derecho en la página, podemos ver la solicitud en la sección de red



Si agregamos cualquier palabra o ruta después del puerto, aún obtendremos la misma respuesta del mismo texto sin formato. Para verlo mejor, agregamos una línea a la función `createServer` para ver la url que envía la solicitud:



```
var server=http.createServer(function (req,res) {  
  console.log(req.url);  
  res.writeHead(200, {'Content-type':'text/plain'});  
  res.end('hola clase');  
});
```

Aprenderemos a controlar la respuesta con diferentes urls por enrutamiento.

2. Sirviendo JSON

Servir JSON al cliente

JSON es una forma de escribir los datos con notación

javascript:https://www.w3schools.com/js/js_json_intro.asp

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>

Es ampliamente utilizado para enviar datos entre servidores y clientes. Por ello, debemos aprender a enviarlos desde nuestro servidor de nodos.

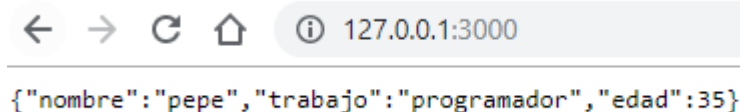
Lo primero es cambiar el tipo de datos que enviamos. También creamos un objeto JSON simple y podemos pensar en hacer algo como esto

```
var server=http.createServer(function (req,res) {  
  console.log(req.url);  
  res.writeHead(200, {'Content-type':'application/json'});  
  var miObj={  
    nombre:'pepe';  
    trabajo:'programador';  
    edad:35;  
  }  
  res.end(miObj);  
});
```

Sin embargo, el método final toma una cadena o un búfer como parámetro y myObj no lo es. Para arreglar esto, lo "stringificamos":

```
var server=http.createServer(function (req,res) {
  console.log(req.url);
  res.writeHead(200, {'Content-type':'application/json'});
  var miObj={
    nombre:'pepe',|
    trabajo:'programador',
    edad:35
  }
  res.end(JSON.stringify(miObj));
});
```

Para convertir una cadena a json usamos JSON.parse (cadena).



A screenshot of a web browser's address bar and content area. The address bar shows the URL '127.0.0.1:3000'. Below it, the browser displays the JSON response: {"nombre":"pepe","trabajo":"programador","edad":35}.

Una solicitud JSON puede provenir de algún código javascript que la crea y luego muestra los datos de cierta manera en el front-end de una página web.

Lo habitual es trabajar con archivos JSON. Podemos combinar los modos asíncrono/sincrónico que vimos anteriormente con los métodos que vemos ahora.

Para trabajar con un archivo JSON:

Leyendo un archivo JSON:

Método 1: Usar el método require: El método más simple para leer un archivo JSON es requerirlo en un archivo node.js usando el método require().
Sintaxis:

```
const data = require('ruta/a/archivo/nombre de archivo');
```

Ejemplo: Cree un archivo users.json en el mismo directorio donde está presente el archivo index.js. Agregue los siguientes datos al archivo json.

archivo de usuarios.json:

```
[
  {
    "nombre": "Juan",
    "edad": 21,
    "idioma": ["JavaScript", "PHP", "Python"]
```

```

},
{
  "nombre": "Smith",
  "edad": 25,
  "idioma": ["PHP", "Ir", "JavaScript"]
}
]

```

Ahora, agregaremos el siguiente código a su archivo index.js.

archivo index.js:

```

// Requerir archivo de usuarios
const usuarios = require("./usuarios");

console.log(usuarios);

```

Ahora, ejecute el archivo usando el comando:

node index.js

Método 2: Usando el módulo fs: También podemos usar el módulo fs de node.js para leer un archivo. El módulo fs devuelve el contenido de un archivo en formato de cadena, por lo que debemos convertirlo a formato JSON utilizando el método integrado JSON.parse().

Agregue el siguiente código en su archivo index.js:

archivo index.js:

```

const fs = require("fs");

// Lee el archivo users.json
fs.readFile("usuarios.json", function(err, datos) {

  // Comprobar si hay errores
  si (err) tirar err;

  // Convirtiendo a JSON
  const usuarios = JSON.parse(datos);

  console.log(usuarios); // Imprimir usuarios
});

```

Ahora ejecute el archivo de nuevo

Escribir en un archivo JSON:

Podemos escribir datos en un archivo JSON usando el módulo node.js fs. Podemos usar el método writeFile para escribir datos en un archivo.

Sintaxis:

`fs.writeFile("nombre de archivo", datos, devolución de llamada);`
Ejemplo: Agregaremos un nuevo usuario al archivo JSON existente, que hemos creado en el ejemplo anterior. Esta tarea se completará en tres pasos:

Lea el archivo utilizando uno de los métodos anteriores.
Agregue los datos usando el método `.push()`.
Escriba los nuevos datos en el archivo utilizando el método `JSON.stringify()` para convertir los datos en una cadena.

```
const fs = require("fs");
```

```
// PASO 1: Lectura del archivo JSON
const usuarios = require("./usuarios");
```

```
// Definiendo nuevo usuario
let usuario = {
  nombre: "Nuevo Usuario",
  edad: 30,
  idioma: ["PHP", "Ir", "JavaScript"]
};
```

```
// PASO 2: Agregar nuevos datos al objeto de los usuarios
usuarios.push(usuario);
```

```
// PASO 3: Escribir en un archivo
fs.writeFile("usuarios.json", JSON.stringify(usuarios), err => {
```

```
// Comprobación de errores
if (err) throw err;
```

```
console.log("Terminó de escribir"); // Éxito
});
Ejecuta el archivo de nuevo
```