Unidad **14**

BackEnd - Node.JS

1 require HTTP
2 writeHead – Crear / Modificar el encabezado HTTP
3 end() para el envío de cadenas en HTML
4 console.log desde Node.
5 Creando un servidor. El módulo HTTP
6 Configuración del puerto de escucha con listen()
7 Usando http.createServer()
8 Análisis de su funcionamiento y detalle de librerías activas.
9 Instalando librerías a Visual Studio
10 Creando un proyecto en Node con NPM
11 Leer Cadenas en consultas
12 El error 404 y el código 200
13 Módulo de archivos.

Agregar un encabezado HTTP

Para que la respuesta del servidor HTTP se muestre como HTML, debe incluir un encabezado HTTP con el tipo de contenido correcto:

Ejemplo

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hola Ñandú!');
  res.end();
}).listen(8080);
```

En nuestro ejemplo, tanto los acentos como la Ñ no se verían bien ya que hay que configurar los charset en el encabezado:

El encabezado debería modificarse y quedar como el siguiente:

res.writeHead(200, {'Content-Type': **'text/html; charset=utf-8'**});

El primer argumento del método res.writeHead() es el código de estado, 200 significa que la respuesta a ejecutar está correcta, el segundo argumento es un objeto que contiene los encabezados de respuesta. Charset, es la elección de un código de juego de caracteres que soporta acentos, Ñ y todos los demás símbolos en español del teclado.

El módulo HTTP integrado

Node.js tiene un módulo incorporado llamado HTTP, que permite a Node.js transferir datos a través del Protocolo de transferencia de hipertexto (HTTP).

Para incluir el módulo HTTP, usamos el método require():

```
var http = require('http');
```

Node.js como servidor web

El módulo HTTP puede crear un servidor HTTP que escucha los puertos del servidor y da una respuesta al cliente.

Utilizamos el método createServer()para crear un servidor HTTP:

Ejemplo

```
var http = require('http');
//crea un objeto server:
http.createServer(function (req, res) {
  res.write('Hola Mundo Node.Js!'); //escribe del lado del cliente
  res.end(); //cierra el canal de envío de datos
}).listen(8080); //el puerto se define en el nro 8080
```

La función pasada al método http.createServer() se ejecutará cuando alguien intente acceder a la computadora en el puerto 8080.

Guardá el código anterior en un archivo llamado "demo_http.js" y podés ejecutar el archivo con la orden:

```
C:\Users\TuNombre>node demo_http.js
```

Si seguiste los pasos en tu computadora, verás el mismo resultado que en el ejemplo cuando en el navegador vayas a la dirección: http://localhost: 8080

Hola Mundo Node.Js!

Leer la cadena de consulta

La función pasada a http.createServer() tiene un argumento req que representa la solicitud del cliente, como un objeto http.IncomingMessage. Este “require” o requerimiento se encarga de enviar las peticiones para que el servidor atienda y responda su solicitud.

El objeto en si, tiene una propiedad llamada "url" que contiene la parte de la URL que viene después del nombre de dominio:

demo_http_url.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

Guarda el código anterior en un archivo llamado "demo_http_url.js" e iniciá el archivo:

Inicie demo_http_url.js:

```
C:\Users\Tunombre>node demo_http_url.js
```

Si seguiste los mismos pasos en tu computadora, deberías ver dos resultados diferentes al abrir estas dos direcciones:

http://localhost:8080/verano

Producirá este resultado:

/verano

http://localhost:8080/invierno

Producirá este resultado:

/invierno

Manejo de errores de archivos con Error: 404

Desde el servidor de Node podemos seleccionar la respuesta que enviaremos al cliente que solicita un requerimiento. Para ello, los códigos de respuesta pueden ser enviados tomando en cuenta la situación de la respuesta.

Por ejemplo:

Si la lectura de un fichero arroja un error podremos avisar al navegador web con el código 404 que indica que la página o el archivo solicitado no existe, o utilizar el código 200 que indica que todo se ejecutó correctamente y la devolución de la petición no conlleva error alguno.

Vamos a crear un archivo Node.js que abra el archivo solicitado y devuelva el contenido al cliente. Si algo sale mal, lanza un error 404:

demo_archivo.js:

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Archivo no encontrado");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Dividir la cadena de consulta

Hay módulos integrados para dividir fácilmente la cadena de consulta en partes legibles, como el módulo URL.

Ejemplo

Divide la cadena de consulta en partes legibles:

```
var http = require('http');
var url = require('url'); realiza la conversión del formato url
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

Guarda el código anterior en un archivo llamado "demo_querystring.js" e ejecutá el archivo:

Inicie demo_querystring.js:

```
C:\Users\TuNombre>node demo_querystring.js
```

La dirección:

http://localhost:8080/?year=2021&month=julio

Producirá este resultado:

2021 Julio

Módulo ‘fs’ del sistema de archivos Node.js.

Parte I

Node.js como servidor de archivos

El módulo del sistema de archivos Node.js te permite trabajar con el sistema de archivos de tu computadora.

Para incluir el módulo del sistema de archivos, hay que utilizar el método require():

```
var fs = require('fs');
```

Uso común del módulo Sistema de archivos:

- Leer archivos
- Crear archivos
- Actualizar archivos
- Borrar archivos
- Cambiar el nombre de los archivos

Leer archivos

El método fs.readFile()se utiliza para leer archivos.

Supongamos que tenemos el siguiente archivo HTML (ubicado en la misma carpeta que Node.js):

index.html

```
<html>
<body>
<h1>Título</h1>
<p>Un párrafo.</p>
</body>
</html>
```

Crearemos un archivo Node.js que lea el archivo HTML y devuelva el contenido:

Ejemplo

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('index.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
```

```
    return res.end();
  });
}).listen(8080);
```

Guardá el código anterior en un archivo llamado "demo_readfile.js" e iniciá el archivo:

Inicie demo_readfile.js:

```
C:\Users\TuNombre>node demo_readfile.js
```

Si has seguido los mismos pasos en tu computadora, verás el mismo resultado que en el ejemplo ingresando en tu navegador: http://localhost:8080

Crear archivos

El módulo del sistema de archivos tiene métodos para crear nuevos archivos:

- fs.appendFile()
- fs.open()
- fs.writeFile()

El método fs.appendFile() agrega contenido especificado a un archivo. Si el archivo no existe, se creará el archivo.

Ejemplo

Cree un nuevo archivo usando el método appendFile ():

```
var fs = require('fs');

fs.appendFile('nuevoarchivo.txt', 'Hola Node!', function (err) {
  if (err) throw err;
  console.log('Archivo Guardado!');
});
```

El método fs.open() toma una "parámetro" como segundo argumento, si el parámetro es "w" , el archivo especificado se abre para escribir. Si el archivo no existe, se crea un archivo vacío:

Ejemplo

Cree un archivo nuevo y vacío usando el método open ():

```
var fs = require('fs');

fs.open('archivo.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Guardado!');
});
```

El método fs.writeFile() reemplaza el archivo y el contenido de éste si existe. Si el archivo no existe, se creará un nuevo archivo con el contenido especificado:

Ejemplo

Crearemos un nuevo archivo usando el método writeFile ():

```
var fs = require('fs');

fs.writeFile('archivo.txt', 'Hola Node!', function (err) {
  if (err) throw err;
  console.log('Guardado!');
});
```

Módulo ‘fs’ del sistema de archivos Node.js.

Parte II

Actualizar archivos

El módulo del sistema de archivos tiene los siguientes métodos para actualizar archivos:

- fs.appendFile()
- fs.writeFile()

El método fs.appendFile()agrega el contenido especificado al final del archivo especificado:

Ejemplo

Incorporar "Este es el nuevo texto." al final del archivo "archivo.txt":

```
var fs = require('fs');

fs.appendFile('archivo.txt', ' Este es el nuevo texto.', function (err) {
  if (err) throw err;
  console.log('Se actualizó el archivo!');
});
```

El método fs.writeFile() reemplaza el archivo y el contenido especificados:

Ejemplo

Reemplazamos el contenido del archivo "archivo.txt":

```
var fs = require('fs');
fs.writeFile('archivo.txt', 'Este es el nuevo texto', function (err) {
  if (err) throw err;
  console.log('Archivo Reemplazado!');
});
```

La diferencia fundamental en este método, es que el archivo no es creado nuevamente, sino que se reemplaza el dato contenido en él.

Borrar archivos

Para eliminar un archivo con el módulo Sistema de archivos, usamos el método fs.unlink().

Ejemplo

Eliminar "archivo.txt":

```
var fs = require('fs');
fs.unlink('archivo.txt', function (err) {
  if (err) throw err;
  console.log('Archivo eliminado!');
});
```

Cambiar el nombre de los archivos

Para cambiar el nombre de un archivo con el módulo Sistema de archivos, utilizá el método fs.rename().

Ejemplo

Cambiamos el nombre de "archivo.txt" a "backup.txt":

```
var fs = require('fs');

fs.rename('archivo.txt', 'backup.txt', function (err) {
  if (err) throw err;
  console.log('Archivo renombrado a backup.txt!');
});
```