TEMA 14. Introducción a Node.js (Actualizado a módulos ES6).

Desarrollo Web en Entorno Cliente.

Profesor: Juan José Gallego García

Índice :

- Introducción.
- Ejemplo ‘Hola mundo’
- Módulos Node.js
- Módulo http
- Módulo fs
- Módulo url
- Gestor de paquetes NPM
- Módulo Formidable (sube archivos)
- Módulo Nodemailer (envía correos)
- Módulo mysql (conexión BBDD)
- Framework Express
- Bibliografía.

Introducción

En la unidad anterior hemos instalado **Node.js** y su gestor de paquetes **npm** para poder trabajar con TypeScript, y aunque **Node.js**, es un entorno de ejecución del lado del servidor, vamos a aprovechar y ver en este tema (brevemente) cómo nos permite hacer aplicaciones completas usando el mismo lenguaje que se usa en el lado del cliente (JS, TS ...).

Node.js es un servidor multiplataforma, de código abierto, asíncrono, basado en el lenguaje de programación ECMAScript y motor V8.

Para ver su funcionamiento, se muestra a continuación el ejemplo '**Hola mundo**', y más adelante estudiaremos los principales módulos y demás elementos que componen el código de una aplicación para **Node.js**.

Ejemplo ‘Hola mundo’

1. Creamos una carpeta nueva para guardar los archivos del proyecto. Nos situamos en ella y lanzamos en la consola de VSCode el comando `npm init` aceptando todo por defecto se creará el archivo de configuración **package.json**.
2. Creamos y guardamos un nuevo archivo **index.js** con el siguiente contenido:

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

3. Lanzamos el archivo **index.js** con el comando: `node index.js` se creará un servidor en el puerto 8080 y responderá con *‘**Hola mundo**’* en su llamada.
4. Nos vamos al navegador y probamos: http://localhost:8080/

*Para no tener que reiniciar el servidor con los cambios que se realicen, instalamos el paquete **nodemon***
`npm install -g nodemon` *ejecutamos con:* `nodemon index.js`

Módulos Node.js

Para hacer uso de cada objeto, **Node.js**, usa la importación de una serie de módulos ya predefinidos, como si fueran librerías, algo similar al **import/export** en JS y TS. Existen muchos tipos de módulos, incluso se pueden crear personalizados, al exportarlos ya estarían preparados para su importación.

Para importar un módulo usamos : require ( ) (módulos CommonJS) ó import (módulos ES6, en este caso hay que añadir al archivo **package.json** -> `"type":"module"` y usar extensión `*.mjs` para los módulos exportados)

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

```
import { createServer } from 'http';
createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hola mundo');
}).listen(8080);
```

En este caso se importa el módulo **http**, como ya vimos al principio del tema, pudiéndose acceder a sus métodos para habilitar un servidor y enviar respuestas al cliente (navegador)

Creamos nuestros propios módulos usando export, lo guardamos con el nombre indicado y a continuación hacemos uso de ellos.

```
// miModulo.mjs
export function suma (a,b) {
   return a+b;
 }
```

```
    // miClase.mjs
  export class Saludo {
    constructor() {}
    hi (){return "Hola"}
  }
```

```
import { createServer } from 'http';
import {suma} from './miModulo.mjs';
import {Saludo}  from './miClase.mjs';
var saluda = new Saludo();
createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write("La suma es: " + suma(3,4));
    res.write(saluda.hi());
    res.end();
}).listen(8080);
```

Es necesario usar ‘ **./** ‘ en la importación del módulo y la extensión ‘**./nombreDeModulo.mjs**’ , para indicar que está en el mismo directorio.

Módulo **http**

Permite habilitar a **Node.js** como servidor y responder a las peticiones desde los clientes.

- createServer ( ) , método para crear servidor y se le pasa una función con dos parámetros (**req** y **res**) , que es ejecutada para gestionar la respuesta al cliente.
- res.writeHead( ), dentro de la función pasada y junto al parámetro **res** establece el tipo correcto de contenido en el encabezado.
- req.url , esta propiedad del primer parámetro, devuelve la parte de 'ruta' url que acompaña al nombre del dominio. Ejecutamos como servidor el ejemplo que viene a continuación y usamos la url en el navegador : http://localhost:8080/directorio

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

Módulo **fs**

El módulo **fs** permite trabajar con el sistema de archivo del servidor, leyendo, creando, renombrando, modificando, o borrando archivos.

- readFile( ), *método que lee un archivo en el servidor.* ***Ej:***
  Creamos un archivo (cualquiera) **demo1.html** en el servidor, haremos una lectura de éste, y lo enviaremos al cliente con el siguiente código **leeHtml.js** que vamos a guardar y ejecutar (con **node**) en el propio servidor.

```
// leeHtml.js
import { createServer } from 'http';
import * as fs from 'fs';
createServer(function (req, res) {
  fs.readFile('demo1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
```

- (**Servidor**) Ejecutamos en consola : node leeHtml.js
- (**Cliente Navegador**) http://localhost:8080/

- writeFile( ), crea o reemplaza si existe un archivo y su contenido.

```
fs.writeFile('prueba1.txt', 'Contenido….', function (err) {
    if (err) throw err;
    console.log('Creado!');
  });
```

- appendFile( ), añade contenido al final del archivo especificado.

```
fs.appendFile('prueba1.txt', '\n más contenido….', function (err) {
  if (err) throw err;
  console.log('Añadido!');
});
```

- unlink( ), borra el archivo especificado.

```
fs.unlink('prueba1.txt', function (err) {
  if (err) throw err;
  console.log('Archivo borrado!');
});
```

- rename( ), renombra un archivo.

```
fs.rename('prueba1.txt', 'prueba2.txt', function (err) {
  if (err) throw err;
  console.log('Archivo renombrado!');
});
```

Cambia el nombre del archivo **prueba1.txt** a **prueba2.txt**

Módulo **url**

Permite extraer información acerca de la partes que forman una **url**

- Método url.parse( ) , devuelve un objeto URL con cada parte de la dirección como propiedades. Alternativa con API actualizada: new URL( )

Ej:

```
import * as url from 'url';
var adr = 'http://localhost:8080/index.html?nombre=Juan&apellido=Gallego';
var q = url.parse(adr, true);
console.log(q.host); //devuelve 'localhost:8080'
console.log(q.pathname); //devuelve '/index.html'
console.log(q.search); //devuelve '?nombre=Juan&apellido=Gallego'
var qdata = q.query; //devuelve: { nombre: 'Juan', apellido: 'Gallego' }
console.log(qdata.apellido); //devuelve 'Gallego'
// Como forma más actualizada se puede usar la nueva API ----------:
var url1=new URL("http://localhost:8080/index.html?nombre=Juan&apellido=Gallego");
console.log(url1.pathname); //devuelve '/index.html'
```

Si se establece el segundo parámetro del método url.parse( ) en true, permite devolver el objeto en la propiedad q.query, si se omite es false, y queda sin definir el objeto.

En el siguiente ejemplo se devuelve el archivo html solicitado en el navegador usando los módulos vistos anteriormente.

```
import * as http from 'http';
import * as fs from 'fs';
//var url=require('url');
http.createServer(function (req, res) {
   // var q = url.parse(req.url, true);
    var q= new URL(`http://`+req.headers.host+req.url);
    var filename = "." + q.pathname;
    fs.readFile(filename, function(err, data) {
        if (err) {
        res.writeHead(404, {'Content-Type': 'text/html'});
        return res.end("404 Not Found");
        }
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.write(data);
        return res.end();
    });
}).listen(8080);
```

http://localhost:8080/dibujo.html

```
<!-- dibujo.html -->
<!DOCTYPE html>
<html>
<body>
<h1>Hola desde Dpto. Dibujo</h1>
</body>
</html>
```

http://localhost:8080/naturales.html

```
<!-- naturales.html -->
<!DOCTYPE html>
<html>
<body>
<h1>Hola desde Dpto. Naturales</h1>
</body>
</html>
```

Gestor de paquetes NPM

Como ya sabemos NPM es un gestor de paquetes disponibles para Node.js que facilita la disponibilidad de módulos o librerías para éste. Aunque Node.js posee paquetes preinstalados disponibles para su uso, existen miles de ellos que se pueden instalar cuando sean necesarios.

A continuación vemos cómo instalar un paquete para convertir a mayúsculas.

- Instalamos el paquete desde consola `npm install upper-case`
- Se creará una carpeta node_modules donde se guardarán todos los nuevos paquetes.
- Usamos el módulo:

```
import { createServer } from 'http';
import { upperCase } from 'upper-case';
createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(upperCase("Hola mundo!"));
  res.end();
}).listen(8080);
```

Módulo Formidable (sube archivos)

Facilita la subida de un archivo al servidor. Instalamos previamente los paquetes : `npm install slugify` , `npm install formidable@v3`

El siguiente ejemplo usa el módulo **formidable** para subir un archivo, copiaremos el ejemplo en un archivo **.js** y lo ejecutaremos con node.
ejemplo subir archivo al servidor

Módulo Nodemailer (envía correos) (IMPORTANTE usa **require()**)

Facilita el envío de correos desde el servidor usando una cuenta de Gmail por ejemplo. Para poder usar la cuenta de Gmail , hay que habilitar el acceso de aplicaciones poco seguras, en: *Gestionar tu cuenta de Google/Seguridad/Acceso de aplicaciones poco seguras (Activar) ó Gestionar tu cuenta de Google/Seguridad/Iniciar sesión en Google/Activar verificación en dos pasos - Contraseñas de aplicaciones (generar passw)*

Instalamos el paquete: `npm install nodemailer`
ejemplo envío correo

Módulo mysql (conexión BBDD)

Con este módulo podemos conectar con una BBDD **Mysql**, crear una nueva o gestionarla. Primero instalamos el paquete necesario: `npm install mysql`

En el siguiente ejemplo se conecta con la BBDD tema10 (creada en Mysql en el tema 10), con el usuario y password indicados en la conexión. Seguidamente se crea una nueva tabla y se inserta un registro.

ejemplo conexión BBDD Mysql

Guardamos como archivo **.js** y ejecutamos con node.

Igualmente podemos conectar y gestionar una BBDD **MongoDB** (NoSQL) instalando el paquete : `npm install mongodb` ver referencias web en Bibliografía.

Framework Express

Es una marco de trabajo minimalista para Node.js que facilita la creación de aplicaciones para Node.js. A continuación se indican los pasos que seguiremos para crear el ejemplo *'Hola mundo'*.

- Creamos y establecemos un directorio de trabajo en la consola de VSCode.
- Ejecutamos `npm init` para crear archivo de configuración por defecto **package.json**, aceptando todo por defecto.
- Instalamos el paquete con `npm install express`
- Creamos un nuevo archivo **app.js** y pegamos el siguiente código.

```
import express from 'express';
var app = express();
app.get('/', function (req, res) {
  res.send('Hola mundo!');
});
app.listen(3000, function () {
  console.log('Ejemplo hola mundo escuchando en el puerto 3000!');
});
```

- Ejecutamos **app.js** con: `node app.js` y abrimos en el navegador http://localhost:3000/

Se puede comprobar , tal y como le hemos dicho al **.js** , que responda en el directorio raíz con 'Hola mundo', si queremos enrutar o direccionar una página podríamos añadir :

```
app.get('/dir1', function (req, res) {
    res.send('<h1>Hola mundo en DIR1!</h1>');
  });
```

Y al ejecutar en el navegador http://localhost:3000/dir1 devolvería el html establecido.

Esto es solo una introducción de este marco, no se pretende conocerlo en profundidad, aún siendo minimalista, es muy flexible y completo.
Ver más información en la web oficial (*Bibliografía*)

En el siguiente ejemplo se utiliza el manejador de ruta **express.Router()** , para devolver el contenido de los archivos html estáticos en función del enrutamiento solicitado por GET (también es posible por POST).

Enlace archivo servidor **.js**

index.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Index.html leído</h1>
</body>
</html>
```

Pag1.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Pag1.html leído</h1>
</body>
</html>
```

Pag2.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Pag2.html leído</h1>
</body>
</html>
```

El manejador de rutas **router** permite exportar el grupo de rutas definidas.

http://localhost:3000/
http://localhost:3000/pag1
http://localhost:3000/pag2

Por último, como alternativa y mejora, escrito por el mismo desarrollador de **Node.js** tenemos la opción de trabajar con Deno, mejora algunos aspectos como la eficiencia, llamadas basadas en promesas, usa más eficientemente los módulos ES6, y otros aspectos que hacen que esta alternativa sea más prometedora … ver siguiente enlace

Bibliografía

- https://www.w3schools.com/nodejs/
- https://expressjs.com/es/
- Deno