Informe sesión de programación 2. Node Js

Jesus Rodolfo Izurieta Veliz

June 21, 2020

Contents

1 Introducción

Enunciado del proyecto

Desarrollo de un servidor web que atienda peticiones de archivos con diferentes formatos (ASCII, JSON, HTML, PDF). Los archivos estarán distribuidos en dos únicos directorios: *pub y /doc). Cuando el servidor reciba una petición GET con la ruta '/doc*', deberá recuperarlo desde el directorio /doc; el mismo procedimiento deberá seguirse con la ruta '/pub'. Por ejemplo: /pub/test.html

El servidor deberá verificar si el archivo solicitado existe antes de recuperarlo, de lo contrario deberá retornar una página comunicando esta condición de error (Codigo 404 – Not Found). Use los módulos "http", "url" y "fs" de Node.

2 Descripción

Para la creación del servidor con NodeJs, usaremos algunos módulos de npm, entre ellos, url y fs; los instalamos con el manejador de módulos npm.

2.1 Módulos importados

2.1.1 Url

El módulo url de node, nos provee utilidades para la resolución de urls. Para esto, cuenta con la clase url, lo que añade atributos que no serían accesibles en el caso de usar únicamente cadenas de texto.

2.1.2 Fs

El módulo fs implementa una interfaz del sistema de archivos del sistema operativo para node, de modo que con ayuda de fs seremos capaces de acceder a directorios y archivos de nuestro servidor.

3 Desarrollo

3.1 Servidor http

Creamos el archivo app.js, donde pondremos el código del servidor, lo primero que incluiremos en el documento serán los módulos importados que usaremos:

```
const http = require('http');
const url  = require('url');
const fs   = require('fs');
const path = require('path');
```

Http es el módulo que nos permitirá recibir y responder a peticiones http. Url contiene utilidades para resolución de urls. Fs nos dará acceso al sistema de archivos del sistema operativo. Path nos provee utilidades para el correcto manejo de direcciones de archivos.

A continuación definimos algunas variables para configurar el servidor:

```
const serverHostname  = 'http://localhost:8080/';
const serverPort = 8080;
```

Nos servirán para definir el nombre del host y el número de puerto de nuestro servidor web. Seguidamente desarrollamos el servidor, donde aceptaremos las peticiones usando el módulo http.

```
http.createServer()
```

La función createServer del módulo http recibe una función con dos parámetros, dos objetos: request y response que contienen la información de la petición y proveen un medio para enviar una respuesta, respectivamente. Esta función nos devolverá un objeto del que podremos llamar a su método listen con el puerto y el nombre del host, para que el servidor escuche peticiones.

```
http.createServer(/* código del servidor */)
    .listen(serverPort, serverHostname);
```

3.2 Respuesta a peticiones de archivos

La función que pasaremos como parámetro de http.createServer contendrá el código para responder a las peticiones realizadas por un cliente como un navegador.

```
function server (req, res) { /* código de servidor */ }
```

Primeramente obtendremos la url y la convertiremos en un objeto de tipo URL, para poder usar los métodos que implementa esta clase. Creamos una nueva instancia de la clase URL, a la que pasamos como parámetros la url obtenida por la petición y el nombre del host del servidor, variable que instanciamos anteriormente.

```
let request_url = new URL(req.url, serverHostname);
```

Usaremos el método join de la clase path, para obtener la dirección local, concatenando el directorio actual ('.') y la dirección del archivo que devolveremos, que corresponde a la dirección de la url de la petición, valor que podremos encontrar en el atributo pathname del objeto URL.

```
let filename = path.join('.', request_url.pathname);
```

Finalmente, con el nombre y dirección del archivo al que se quiere acceder, usamos el módulo fs para obtener el archivo (si existe), Para esto usaremos el método readFile de fs, que recibe como parámetros el nombre del archivo, el tipo de codificación, y un callback, una función que nos dará acceso a dos parámetros: error, nos devolverá un error si no se puede obtener o abrir el archivo, y data, que nos devolverá el contenido del archivo si este fue encontrado y pudo ser accedido sin problemas.

```
fs.readFile(filename, 'utf-8', function(err, data) {
    if(err){
        res.setHeader('Content-Type', 'application/json');
        res.statusCode = 404;
        let message = {
            code: "404: Not Found",
            message: "No se encuentra el archivo " + filename
        }
        res.end(JSON.stringify(message));
    }
    else {
        res.end(data);
    }
});
```

En caso de que no haya errores en la lectura del archivo, el parámetro err será nulo, por lo que preguntamos si es un valor definido en un if. Si 'error' no es nulo, responderemos con un mensaje de error en formato json, para esto, definimos la cabecera de la respuesta como 'application/json' y el statusCode a 404

```
res.setHeader('Content-Type', 'application/json');
res.statusCode = 404;
let message = {
```

```
    code: "404: Not Found",
    message: "No se encuentra el archivo " + filename
}
res.end(JSON.stringify(message));
```

A continuación creamos un objeto message que contendrá el código de error 404 y un mensaje en texto que será legible por la persona que acceda al servidor. Enviaremos este mensaje como parámetro de la función end del objeto response, no sin antes convertirlo en una cadena con la función stringify del módulo JSON.

Finalemnte en caso de que el objeto sea encontrado y pueda ser leído, enviamos su contenido, almacenado en la variable data.

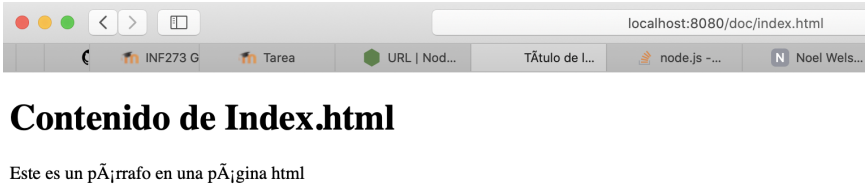
4 Observaciones

Se debe implementar algún mecanismo de seguridad en este servidor, ya que al responder a cualquier url en la petición, cualquier archivo es accesible, incluyendo código fuente del servidor o cualquier otro archivo dentro del mismo directorio.

5 Pruebas

Usaremos un navegador para probar el servidor, incluiremos distintos formatos de archivos y directorios.

Creamos un archivo index.html en el directorio doc y realizando la petición `http://localhost:8080/doc/index.html` obtenemos:



De igual manera, con un archivo de texto, obtenemos el contenido de este: