NODE.JS y MONGODB
CLASE 2 : FS Y EXPRESS

FILESYSTEM

El módulo de FileSystem (fs) nos permite trabajar con el sistema de archivos del sistema operativo de la máquina. Tareas comunes de este módulo incluyen pero no se limitan a :

- Crear Archivos
- Borrar Archivos
- Editar Archivos
- Renombrar Archivos

```
fs.stats(String URL, Function callback(Error error,Object stats))
```

Antes de comenzar a trabajar con un archivo es siempre conveniente comprobar el estado del archivo respecto del servidor y si todo se encuentra bien procedemos a la tarea final sin errores. Este método toma como parametros :

- URL : Un string de la URI ubicación del archivo o fichero en cuestión
- Callback : Una función asincrónica que se ejecuta cuando toda la información del archivo se recibió. La misma puede tomar a su vez dos parámetros, un error y un objeto representando toda la información del archivo o fichero

Método / Propiedad	Descripción
stats.isFile()	Indica si la URI es un archivo
stats.isDirectory()	Indica si la URI es un directorio
stats.mode	Indica el tipo y modo de archivo()
stats.size	Indica el tamaño del archivo en bytes
stats.atimeMs	El tiempo en milisegundos de la última vez que el

	archivo fue accedido
stats.ctimeMs	El tiempo en milisegundos de la última vez que el archivo cambió su status
stats.mtimesMs	El tiempo en milisegundos de la última vez que el archivo fue modificado
stats.birthtimeMs	El tiempo en milisegundos de la creación del archivo

Una vez que obtenemos información necesaria del archivo podemos proceder a abrirlo y saber qué operaciones podemos hacer con él dado que nos enteramos anteriormente si era un directorio o un archivo.

```
fs.open(path, flags[, mode], callback)
```

El método open nos permite abrir un archivo de manera asincrónica. También nos permite configurar el modo de un archivo si este está siendo creado en ese momento. El callback que toma es una función que se ejecuta cuando el archivo se abrió y recibe dos parámetros, un error y un fd puntero del archivo abierto en cuestión.

```
fs.readdir(path[, options], callback)
```

El método readdir nos permite abrir un directorio de manera asincrónica. El segundo parámetro opcional nos permite configurar el encoding de los archivos que vamos a recibir y el callback es una función que se ejecuta cuando el directorio fue abierto y recibe dos parámetros, un error un un listado de archivos.

```
fs.readFile(path[, options], callback)
```

El método readFile nos permite leer un archivo de manera asincrónica. El mismo toma como parámetro alguna referencia del archivo, como su ruta, un Buffer, un URL ó un fd. El parámetro options nos permite configurar el encoding ó el flag en el que queremos abrir el archivo y un callback que es una función que se ejecuta

cuando el archivo fue abierto y recibe dos parámetros, un error y el dato del archivo.

NPM

NPM (Node Package Manager) es justamente como su nombre lo dice, el manager de paquetes de Node.js . En realidad contempla hoy en día no solo paquetes útiles para desarrollo en Node.js sino que nos da prácticamente todos los paquetes de Javascript. Es uno de los repositorios de librerias, plugins y frameworks más grandes de todo el mundo.

Con él podemos no sólo instalar paquetes a nivel local por proyecto sino que también instalar paquetes de manera global a nivel “sistema operativo” y ejecutarlos a través de Node.js como un programa más de nuestra PC. Por sobre todo, también nos sirve para iniciar, configurar y mantener al día nuestros proyectos gracias a una serie de funcionalidades para descargar o actualizar las dependencias que nuestro proyecto requiera, pudiendo compartir toda esta información en un solo archivo config.json entre todo un grupo de trabajo.

Vamos a intentar instalar y usar uno de los frameworks más conocidos y rápidos de Node.js para configuración básica del servidor y sus rutas : Express

EXPRESS

Instalación

La manera de instalarlo, al igual que con cualquier otro paquete de Node.js que queramos de manera local en nuestro proyecto, será :

```
npm install express --save
```

Uso

Podemos rápidamente iniciar un servidor con Express con las siguientes líneas :

```
const express = require('express')
const app = express()
app.get('/', (req, res) => res.send('Hello World!'))
app.listen(3000, () => {
    console.log('Example app listening on port 3000!')
})
```

Si nos detenemos a leer cuidadosamente nos podemos dar cuenta que primero estamos incluyendo el módulo de Express dentro de una constante express. Luego iniciamos una instancia de ese módulo ejecutándose. A continuación configuramos una ruta posible como punto de acceso a nuestra aplicación la cual a su vez tiene un callback con un error y una respuesta asociados. Por último, abrimos un puerto para que el servidor esté escuchando solicitudes entrantes y pueda responderles.

Routing Básico

La sintaxis básica de una ruta cualquiera es:

```
app.METHOD(PATH, HANDLER)
```

De esta forma podemos crear cualquier tipo de entrada y hasta encadenarse entre ellas si apuntan a la misma URI. El handler es una función callback que nos devuelve un objeto para la solicitud(request) y la respuesta(response) respectivamente. Por lo tanto podríamos hacer :

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

Ó

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book')
  })
  .post(function (req, res) {
    res.send('Add a book')
  })
  .put(function (req, res) {
    res.send('Update the book')
  })
```

La respuesta contiene varios métodos de envío de información al cliente. Por ejemplo :

Método / Propiedad	Descripción
response.download()	Muestra una notificación para descargar un archivo
response.end()	Finaliza el proceso de respuesta
response.json()	Envía una respuesta JSON
response.jsonp()	Envía una respuesta JSON con soporte para JSONP
response.redirect()	Redirige la respuesta
response.render()	Le hace render a un template de vista
response.send()	Envía respuestas de varios tipos
response.sendFile()	Envía un archivo como un stream de octetos
response.sendStatus()	Configura el código de la respuesta y lo envía en el cuerpo como string

De este modo también podemos utilizar cookies para el control del estado del cliente :

```
res.cookie(name, value [, options])
```

Donde el parámetro options es un objeto que puede tener las siguientes propiedades :

Método / Propiedad	Descripción
domain	El domínio de la cookie. Defaults a el dominio de la aplicación
encode	Una función sincrónica usada para el encoding de la cookie. Defaults a enodeURIComponent
expires	Fecha de expiración de la cooke en GTM. Si no es especificada o 0(cero) a misma es una cookie de sesión
httpOnly	Si se encuentra activa, la cookie solo es accesible por el servidor
maxAge	Tiempo en número para determinar la vida de la cookie relativa a la fecha actual en milisegundos
path	La ruta de la cookie. Defaults a '/'
secure	Hace que la cookie sea solo accesible por HTTPS
signed	Indica si la cookie tiene que ser firmada
sameSite	El valor del atributo sameSite de Set-Cookie

1. *https://www.w3schools.com/nodejs/nodejs_filesystem.asp*
2. *https://nodejs.org/api/fs.html#fs_class_fs_stats*
3. *https://nodejs.org/api/fs.html#fs_fs_open_path_flags_mode_callback*
4. *https://nodejs.org/api/fs.html#fs_fs_readdir_path_options_callback*
5. *https://nodejs.org/api/fs.html#fs_fs_readfile_path_options_callback*
6. *http://expressjs.com/*