TALLER 03 – NODEJS

Este es un taller de introducción a nodejs, recuerda que nodejs es un entorno que permite ejecutar aplicaciones javascript, con nodejs podemos ejecutar javascript del lado del servidor y sin necesidad de el navegador.

Las aplicaciones nodejs están orientadas a manejar eventos asíncronos, con nodejs es posible que construyamos aplicaciones de escritorio y otro tipo de aplicaciones que puedan ser escalables en la red.

A continuación, vamos a desarrollar los conceptos básicos de node.js a través de la construcción de varios ejemplos.

1. Requerimientos técnicos

A continuación, describimos los requerimientos técnicos para poder iniciar a programar en Node.js

- Instalación de nodejs
    - Se puede obtener desde el sitio web https://nodejs.org/es/
    - Este es multiplataforma, así que lo puedes instalar en diferentes sistemas operativos
    - Para verificar la instalación de nodejs puedes ejecutar en la consola de tu sistema operativo el comando: **node -v**
        - Si se encuentra instalado deberá aparecer en la consola la versión de node que se instaló
    - Verificar que también se instalo correctamente el gestor de paquetes NPM, este se instala junto con el mismo instalador del Node.js. Para verificar que esta correctamente instalado ejecutar el comando: **npm -v**
        - Si se encuentra instalado deberá aparecer en la consola la versión de npm que se instaló
- Editor de código o IDE
    - Se recomienda Visual Studio Code
    - Puedes usar el que te gusté
- Es importante que conozcas sobre el lenguaje JavaScript
- Es importante entender el protocolo HTTP

2. Hello Wolrd

Para construir el primer ejemplo de código debemos seguir los siguientes pasos

- PASO 01 – Crear un archivo de tipo javascript, para esto crea un archivo que se llame helloworld.js
- PASO 02 – Dentro del archivo escribir la siguiente línea que va a escribir en consola un valor

```
console.log('---HELLO WORLD---');
```

- PASO 03 – En la consola del sistema operativo, vamos a la ubicación del archivo que creamos y en esta ubicación ejecutamos el comando: **node helloworld.js**

Como resultado tenemos que se imprimió en la consola el texto que adicionamos en el script.

3. Proyecto calculadora

3.1. Calculadora básica con Javascript ejecutada con node

A continuación, vamos a crear un pequeño proyecto en el cual hagamos las operaciones matemáticas básicas

- PASO 01 – Crear una carpeta para el proyecto
- PASO 02 – Dentro de la carpeta crearemos un archivo calculator.js
- PASO 03 – En el archivo calculator.js vamos a crear las funciones para cada una de las operaciones matemáticas que queremos construir, para esto con javascript construimos las funciones.

```
function add (number1, number2) {
    return number1 + number2;
}
function substract (number1, number2) {
    return number1 - number2;
}
function multiply (number1, number2) {
    return number1 * number2;
}
function divide (number1, number2) {
    if (number2 == 0) {
        console.log ('MATH ERROR. You can´t divide by zero ');
    } else {
        return number1 / number2;
    }
}
```

- PASO 04 – Después de escribir las funciones, al final del archivo colocaremos el llamado a estas para probarlas

```
console.log (add(1,0));
console.log (divide(1,0));
console.log (multiply(1,0));
console.log (substract(1,0));
```

- PASO 05- Ejecutar en la consola el archivo con el comando node **helloworld.js**

Como resultado se va a ver en pantalla el resultado de las operaciones matemáticas que se realizaron.

3.2. Módulos en nodejs

Los módulos en nodejs me permiten exportar funciones y objetos que pueden ser usados por diferentes archivos y otros módulos para reutilizar código

- PASO 1 – En la misma carpeta del proyecto anterior crear un archivo llamado index.js
- PASO 2 – En el archivo calculator.js eliminaremos las líneas que imprimían y llamaban las funciones. Solamente dejaremos las funciones matemáticas que creamos.
- PASO 3 – En el archivo index.js vamos a usar las funciones del archivo calculator.js, para esto debemos importar el archivo para que las funciones se puedan usar, para importarlo vamos a crear una constante y en esta vamos a indicar que se requiere el archivo que contiene las funciones.

```
const calc = require ('./calculator.js');
```

- PASO 4 – Si intentamos hacer el llamado a la función desde index.js así hayamos importado otro script. Esto se debe a que todavía no hemos exportado en el otro modulo las funciones que se van a utilizar, para esto hay varias formas de hacerlo:
    - La primera forma es, exportar las funciones una a una directamente, esto se hace de la siguiente manera en el archivo calculator.js

```
exports.add = add;
exports.divide = divide;
exports.multiply = multiply;
exports.substract = substract;
```

Para probar adicionamos el siguiente código en el index.js, en este se puede evidenciar que la constante calc se esta usando para llamar las funciones exportadas.

```
console.log (calc.add(1,0));
```

```
console.log (calc.divide(1,0));
console.log (calc.multiply(1,0));
console.log (calc.substract(1,0));
```

- La segunda forma es, exportar un objeto en el módulo con diferentes propiedades, para esto en el archivo calculator.js hacemos lo siguiente:
  - Creamos un objeto constante al cual como propiedades le vamos adicionar las funciones creadas

    ```
    const calcObj = {};
    ```
  - Ahora adicionamos las funciones creada como propiedades del objeto anterior

    ```
    calcObj.add = add;
    calcObj.divide = divide;
    calcObj.multiply = multiply;
    calcObj.substract = substract;
    ```
  - Luego exportamos el objeto del modulo

    ```
    module.exports = calcObj;
    ```
  - Y después en el index.js podemos probar que se esta exportando y ejecutar las funciones, para esto usemos este código, probemos y analicemos el resultado:

    ```
    const calc = require ('./calculator.js');

    console.log (calc);
    console.log (calc.add(1,0));
    console.log (calc.divide(1,0));
    console.log (calc.multiply(1,0));
    console.log (calc.substract(1,0));
    ```

De esta forma acabamos de aprender como usar node y los módulos de una forma básica, a continuación exploraremos otros conceptos que se usan en aplicaciones node.js.

4. Uso de módulos preinstalados de Node.js

Tal como en la sección anterior construimos un modulo con el cual podíamos realizar cálculos matemáticos básicos, de la misma forma Node.js dentro de su instalación ya tiene algunos módulos preconstruidos que pueden ser usados para construir aplicaciones, estos módulos los podemos consultar en la documentación oficial de node de acuerdo con la versión de node que tengamos.

A continuación, vamos a dar ejemplo de algunos módulos preconstruidos de node y su forma de usarlos, vamos usar el modulo Crypto y el modulo OS. El modulo Crypto permite realizar cifrado de datos y el modulo FilsSystem.

4.1. Uso del modulo Crypto

- Para usar este modulo, lo primero que debemos hacer es importarlo

```
const crypto = require('crypto');
```

- Y con este modulo vamos a generar un hash como ejemplo del uso del modulo, para esto usaremos las funciones createHmac, update, digest si queremos ver más información sobre estas funciones y el modulo podemos refererirnos a la documentación (https://nodejs.org/docs/latest-v10.x/api/crypto.html#crypto_hmac_digest_encoding)
- Para generar el has usaremos el siguiente código

```
const secret = 'helloworld';
const hash = crypto.createHmac('sha256', secret)
                   .update('I love web development')
                   .digest('hex');
```

- El resultado obtenido lo vamos a ver imprimiéndolo en consola

```
console.log(hash);
```

4.2. Uso del modulo File System

- Para hacer uso de este modulo primero debemos importarlo

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

- Con este modulo vamos a escribir en un archivo, usando la función writeFile, esta función recibe como parámetro la ruta de ubicación del archivo,el texto que se va a escribir, y una función de callback la cual se disparara cuando se finalice de escribir en el archivo

```
let filePath = './mytextfile.txt';
let data = 'Hello this is my first file writed with nodejs';
fs.writeFile (filePath, data , function (err){
    if (err) {
        console.log ('ERROR', err);
    }
    console.log ('The file has been succesfully created!');
});
console.log ("END OF SCRIPT");
```

- Ejecutar el código con node y verificar lo que sucede, es importante entender que la función writeFile del modulo file system funciona de forma asíncrona, por esto el texto “END OD SCRIPT” aparecerá primero que el texto “The file has been succesfully created!”. Verificar que el archivo de texto se creo y que tiene el contenido que le escribimos.

5. Creación de un servidor para manejar peticiones HTTP

Con node podemos crear una aplicación de servidor usando el protocolo HTTP, este protocolo esta basado en el paradigma cliente/servidor, para realizar esta tarea vamos a utilizar el modulo HTTP de node.js, este es un modulo preconstruido.

5.1. Creación del servidor

- PASO 1 – Crear una carpeta
- PASO 2 – Crear un archivo index.js
- PASO3 – Importar el modulo HTTP

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

- PASO 4 – Crear el servidor

  Para crear el servidor usamos la función createServer del modulo http.

```
let server = http.createServer (function (request, response) {
    response.write ('<h1>HELLO WORLD</h1>');
    response.end();

});
```

- PASO5 – Definir el puerto para que el servidor inicie la escucha a través de este.

```
server.listen (3000);
```

- PASO 6 – Ejecutar con node el archivo y en el navegador ingresar a la ruta localhost:3000

6. Gestor de paquetes NPM

NPM es el gestor de paquetes que usa node.js, este es útil para poder usar módulos que ya han sido construidos por otros desarrolladores o empresas, esto nos permite el uso de frameworks para poder extender la funcionalidad de node y que no todo el trabajo lo hagamos desde javascript puro nosotros mismos.

NPM es un gestor de paquetes de javascript el cual tiene algunos repositorios en los cuales se almacenan muchos de los módulos existentes para poder realizar aplicaciones con mayores funcionalidades.

Con npm podemos administrar los paquetes de dependencias que tienen nuestros proyectos y de esta forma poderlos ejecutar desde cualquier otra maquina.

6.1. Creación de un package.json

NPM funciona a través de un archivo llamado package.json en el cual se almacenan las configuraciones del proyecto con sus dependencias.

- PASO 1 – Crear una carpeta nueva para un proyecto
- PASO 2 – Dentro de la consola del sistema ubicarse en la ruta de la carpeta y ejecutar el comando **“npm init”** y llenar la información solicitada, en la carpeta se debe haber creado un archivo package.json
- PASO 3 – Crear un archivo index.js
- PASO 4 – Dentro del index.js creamos un server de node y que responda un simple texto en el navegador
- PASO 5 – Vamos a instalar un nuevo modulo externo, para esto vamos a usar npm, en este caso vamos a usar uno que se llama boxen para crear cajas en la consola, su documentación la podemos encontrar en https://www.npmjs.com/package/boxen
- PASO 6 - Instlar el paquete de boxen, esto lo hacemos ejecutando el comando **“npm install boxen”,** al correr este comando, npm instala el modulo junto con sus dependencias y adiciona la dependencia de boxen al archivo package.json de nuestro proyecto.
- PASO 7 – Como ya esta instalado boxen ahora lo podemos usar, para esto debemos importarlo en nuestro archivo index.js

```
const boxen = require('boxen');
```

- PASO 8 – Ahora adicionaremos dentro de la función callback de la función listen del server eel siguiente código

```
    console.log(boxen('SERVER START ON PORT 3000', {
        padding: 1,
        borderColor: 'cyan'
```

```
}));
```

- PASO 9 – Ejecutamos el archivo index.js y observamos la consola, en esta debe aparece la caja renderizada con boxen