Node JS:

es un entorno que nos permite ejecutar javascript sin necesidad de un navegador web

características:

podemos instalarlo y ejecutarlo en multipls sistemas operativos

permite interactuar con el mismo sistema operativo

## Caracteristicas:

* Open source
* Multiplataforma
* Basado en el motor v8 de Google.
* Motor v8:
* Es un motor de JavaScript desarrollado por el chromium Project para Google Chrome.
* Es escalable
* Es de tiempo real

OBSERVACIONES:

EN NODE NO EXISTE:

* EL OBJETO WINDOW , HAY ALGO SIMILAR QUE ES GLOBAL
* EL DOM

# como ejecutar Js desde node:

simplemente basta con abrir la terminal en estemos e indicar el comando:

node + “file\_name”

*ejemplo:*

*si tenemos un archivo *

*con la siguiente función.*

*function* saludar(*nombre*){return(`Hola ${nombre} como estas?`);}

console.log(saludar("pepe"));

*escribiríamos esto por consola*

**

*Y obtendríamos esto:*

**

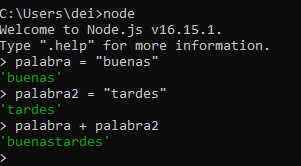
## desde el cmd:

para empezar podemos consultar su versión asi:

node -v



Si escribimos solo “node” podremos usar JS en el “cmd”



## desde vs code:

simplemente abrimos la terminal y podemos consultar de igual forma su versión:



Devuelve esto:



Si quisiéramos trabajar con un archivo especifico:

Sintaxis:



Ejemplo:

Supongamos que tenemos este archivo:



Lo ejecutaríamos asi:



Si tuviese algo como esto:

num1 *=* 3;

num2 *=* 4;

console.log(num1*+*num2);

al ejecutar el comando:

 o 

Obtenemos:



Muy simple.

# Npx:

PERMITE EJECUTAR MODULOS SIN TENER QUE INSTALARLOS, ES COMO UNA EJECUCION VOLATIL QUE BUSCA EL MODULO ESPECIFICO HACE USO DE EL Y LUEGO TERMINA EL PROCESO, SIN INSTALAR NADA.

# MODULOS:

Los modulos son funcionalidades con uno o varios archivos JavaScript, los cuales sirven para utilizarse y reutilizarse en una aplicación.

Supongamos que tenemos un programa muy grande, es inconveniente tenerlo todo en un solo archivo, nosotros podemos dividir varias funcionalidades en módulos y luego comunicarlos entre si , de esta forma también es mas fácil identificar problemas ya que ocurrirían en un modulo en particular y no en todo un archivo, el mantenimiento asi es mucho mas fácil,

Los modulos se pueden importar y exportar en otros modulos

# Exportar e Importar:

Para hacer uso de un modulo en otro modulo, tenemos que exportarlo del primero e importarlo al segundo

## Exportar/Exports:

Supongamos el siguiente archivo:



*function* sumar(*n1*,*n2*){return n1+n2}

*function* resta(*n1*,*n2*){return n1-n2}

*function* multiplicacion(*n1*,*n2*){return n1\*n2}

*function* division(*n1*,*n2*){return n1/n2}

*sintaxis:*

module.exports.name\_module = function\_name;

\*\*module.exports es un objeto\*\*

*Ejemplo 1:*

module.exports.suma = sumar;

module.exports.resta = restar;

module.exports.multiplicacion = multiplicar;

module.exports.division = dividir;

*Ejemplo 2:*

module.exports = {

    suma : sumar,

    resta : restar,

    multiplicacion : multiplicar,

    division : dividir

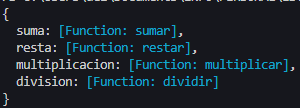
}

\*cualquiera de los dos funciona igual\*

*de echo si vemos esto por consola:*

console.log(module.exports);

**

**

*Podemos observar como se listan las funciones, contenidas en un objeto.*

## Importación/require:

Como la palabra import y export ya existen definidas , se usa la palabra require y se habla de “incluir” en vez de “importar” un archivo, pero igualmente es una importación.

*sintaxis:*

*let/var/const* variable\_name = require("./file\_name.js");

\*por convención se usa *const* \*

*Ejemplo:*

*const* matematicas = require("./operaciones.js");

console.log(matematicas.suma(8,2));

console.log(matematicas.resta(8,2));

console.log(matematicas.division(8,2));

console.log(matematicas.multiplicacion(8,2));



**

# DESESTRUCTURACION:

Ya que los módulos ocasionalmente contendrán muchísimas funciones, objetos y demás, podemos definir que funcionalidades queremos de cada módulo, para así ahorrar recursos y no importar todo el módulo innecesariamente.

Supongamos que tenemos un módulo con varias operaciones matemáticas, pero solo queremos las dos últimas.

**

*function* sumar(*n1*,*n2*){return n1+n2}

*function* restar(*n1*,*n2*){return n1-n2}

*function* multiplicar(*n1*,*n2*){return n1\*n2}

*function* dividir(*n1*,*n2*){return n1/n2}

*function* potenciar(*n1*,*n2*){return n1\*\*n2}

*function* raiz\_cuadrada(*n1*){return Math.sqrt(n1)}



module.exports = {

    suma : sumar,

    resta : restar,

    multiplicacion : multiplicar,

    division : dividir,

    potencia: potenciar,

    raiz : raiz\_cuadrada

}

*Sintaxis:*

*const* { export\_name\_1 ,.., export\_name\_N} = require("./file\_name.js");

*ejemplo:*

*const* { potencia ,raiz } = require("./operaciones.js");

console.log(potencia(2,4));

console.log(raiz(100));

**

**

Una característica interesante es que no tenemos que ingresar el nombre del modulo y podemos directamente usar el nombre de las funciones.

# Modulos Built-in(incorporados):

Estos modulos ya Vienen incorporados con node, y nos permiten utilizar funciones básicas de node:

## CONSOLE:

El modulo console esta incorporado por defecto en node y si, es exactamente el mismo que usamos con JavaScript.

Console.log:

Console.warn

Console.error

## PROCESS:

Provee información sobre el proceso de node que se este ejecutando y tambien da sierto control sobre el mismo

Algo importante del modulo process es que se pueden pasarle parámetros conjunto a un archivo.

TIMERS:



# CLI:

La interfaz de línea de comandos o interfaz de línea de órdenes es un tipo de interfaz de usuario de computadora que permite a los usuarios dar instrucciones a algún programa informático o al sistema operativo por medio de una línea de texto simplesimbolo

\*El CLI oficial de node es NPM\*

# NPM:

Es la colección de paquetes de software más grande del mundo , estos paquetes se pueden usar con NodeJs

##### Paquetes:

Los paquetes son directorios que están escritos en formato Json

Estos paquetes pueden publicarse dentro de la colección de NPM.

##### Módulos:

Es cualquier archivo o directorio en el directorio “node\_modules”, esta carpeta se crea automáticamente al trabajar con npm.

Los modulos pueden ser importados con el comando require()

Todos los paquetes que instalemos o suemos para nuestro proyecto se van a guardar en la carpeta “node\_modules”

Los modulos que contengan un archivo tipo “package.json” son considerados paquetes. El resto no.

##### Dependencia:

Paquete que otro paquete necesita para funcionar correctamente.

## COMANDOS DE NPM

##### Npm init:

Para iniciar nuestra configuración de paquete

##### Npm init --yes:

Crea un package.json por default.

# JSON:

JavaScript Object Notation

JSON es un formato de texto sencillo usado para almacenar y transportar informacion.

Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.

## Package.json

## Package-lock.json:

Se genera automáticamente cuando npm modifica el árbol de la carpeta node modules

### Características:

* Los datos se representan como pares: clave-valor, llamadas propiedades
* Estas propiedades pueden estar rodeadas por:

corchetes o llaves

[] o {}

* Están separados por comas a excepción del último elemento.
* Puede contener:
  + Strings
  + Números
  + Arreglos
  + Booleanos
  + Objetos

Un paquete JSON se puede convertir a un objeto en JavaSript

Y un archivo JS puede obtener la información del formato JSON.

*JSON.parse()*

Permite convertir de JSON a JS

# BANDERAS-FLAGS:

Algunas veces escribimos códigos poniendo “-g” o –“algo”

Las cuales sirven para indicar cosas especificas por ejemplo:

“-g” indica que algo va a accionarse de forma global ya sea un listado una instalación o algo mas.

“-d” indica que un paquete es dependencia de desarrollo.

# Eventos en NodeJS:

Un evento se desata cuando una acción es realizada en una aplicación,

Puede ser, mover el mouse, pulsar una tecla, scrollear, etc…

NodeJs está basado en eventos asíncronos.

# PAQUETES EXTERNOS:

En el mundo hay muchos creadores de paquetes que los comparten en la librería npm para que cualquier persona pueda hacer uso de ellos, tienen diferentes funcionalidades y objetivos.

Puede ser que algunos te sean de utilidad, o de inspiración por eso es importante saber como usarlos.

# Express:

Express.js o simplemente Express, es un entorno de trabajo para aplicaciones web para el programario Node.js, de código abierto y con licencia MIT. Se utiliza para desarrollar aplicaciones web y APIs. El autor original es TJ Holowaychuk y la primera versión se lanzó el 2010.

# EVENTOS:

Los Eventos ocurren cuando nosotros interactuamos de cierta forma con la interfaz y los perfifericos es decir, cuando pulsamos una tecla, movemos el mouse, clickeamos, scrolleamos, si estamos inactivos sin hacer nada, etc…

NodeJS posee un módulo para trabajar con eventos

# MODULO EVENTS:

Permite crear eventos, reaccionar a ellos, emitirlos y determinar que acciones van a suceder luego de que ocurran

# Modulo HTTP:

Le permite a nodeJS transmitir información mediante el protocolo HTTP.

## Metodo createServer:

Este método creara un servidor

*const* servidor\_nuevo = modulo\_http.createServer((*req*,*res*)*=>*{

*res*.end("Respuesta de servidor"); */// end Define lo que pasa al final de la creacion del servidor*

});

### Listen:

Sintaxis:

Nombre\_Servidor.listen(puerto, FUNCION);

Permite escuchar un servidor.

*//\_ listen:*

*//+ Determina el puerto y lo que devolvera el servidor cuando este funcionando*

servidor\_nuevo.listen(3200, ()*=>*{

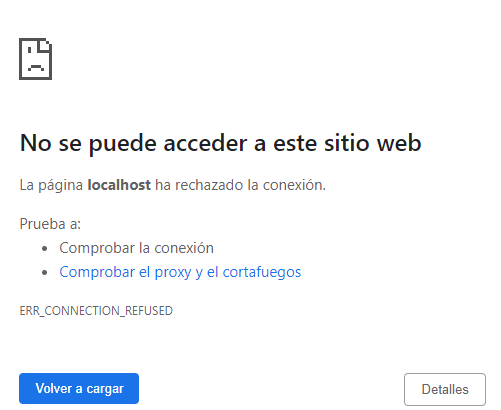
  console.log("el servidor funciona...");

});

Para nosotros acceder a este servidor tenemos que navegar por el navegador (válgala redundancia) hasta el host indicado:

localhost:3200

veremos esto



Porque aun no “montamos” nuestro servidor con node.

Para eso tenemos que estar seguros de que nuestro terminal esta en la misma ruta que contiene el archivo, y luego ejecutar el archivo con node:

**

Si todo salió bien, veremos el mensaje que nosotros definimos anteriormente:



Y tambien podremos consultar la pagina en el navegador con la respuesta que nosotros indicamos:



Claramente nuestra respuesta en este caso es simplemente un texto, pero podría ser un archivo html, que a su vez este realcionado a varios archivos y asi sucesivamente.

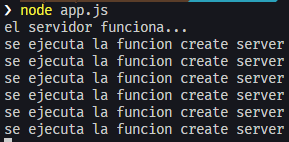
La función contenida en el metodo createServer, se va a ejecutar CADA VEZ QUE CARGUEMOS LA PAGINA.

*const* servidor\_nuevo = modulo\_http.createServer((*req*,*res*)*=>*{

    console.log("se ejecuta la funcion create server");

*res*.end("Respuesta de servidor"); */// end Define lo que pasa al final de la creacion del servidor*

});



A DIFERENCIA DEL PRIMER MENSAJE ESTE SE MUESTRA SOLO LA PRIMERA VEZ QUE LA PAGINA CARGA EFECTIVAMENTE BIEN

## req y res:

son los parámetros que recibe el método createServer.

Req y res: son objetos

Los mismos contienen un montón de información

*const* modulo\_http = require('http');

*const* PUERTO = 3100;

//$ Accediendo a Las propiedades del objeto req:

*const* servidor = modulo\_http.createServer((*req*,*res*)*=>*{

  console.log(*req*.url);

  console.log(*req*.method);

});

servidor.listen(PUERTO, ()*=>*{console.log(`Servidor escuchando en el puerto ${PUERTO}`);})



//$ Accediendo a Las propiedades del objeto res:

*const* servidor = modulo\_http.createServer((*req*,*res*)*=>*{console.log(*res*.statusCode);});

servidor.listen(PUERTO, ()*=>*{console.log(`Servidor escuchando en el puerto ${PUERTO}`);})



Set header:

Permite indicar las cabezeras:

Get headers:

Permite obtener las cabezeras.

*//, setienado y obteniendo headers:*

*const* servidor = modulo\_http.createServer((*req*,*res*)*=>*{

*res*.setHeader('content-type' , 'application/json');

    console.log(*res*.getHeaders);

  });

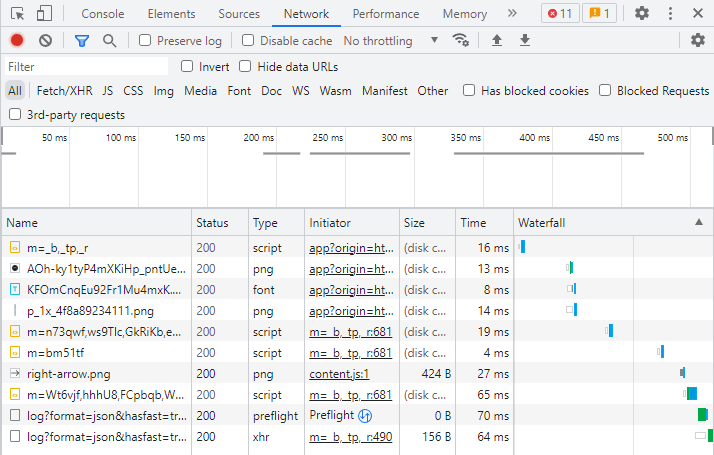
  servidor.listen(PUERTO, ()*=>*{console.log(`Servidor escuchando en el puerto ${PUERTO}`);})



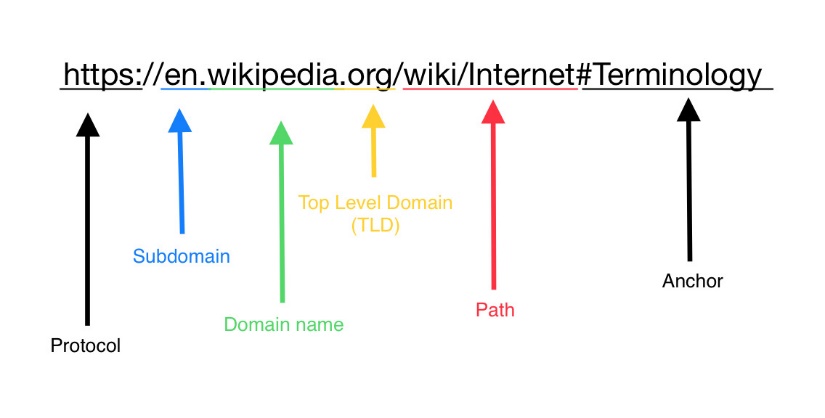
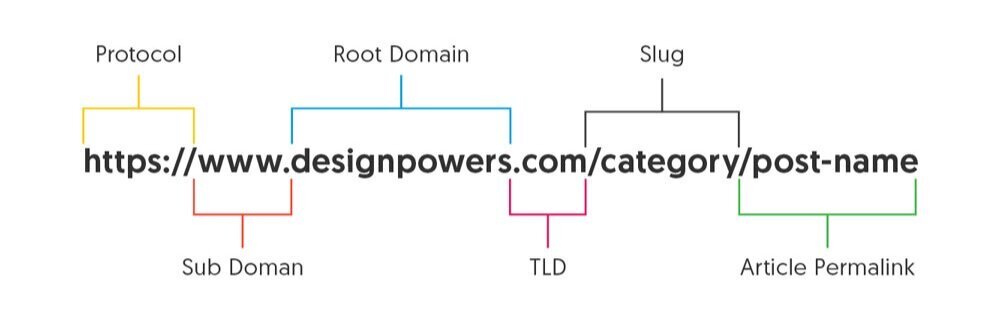
CONSULTAR CODIGOS DE ESTADO

Basta con ir a la pestaña network del inspector en cada pagina web que visites y podras corroborar como es el código de estado de cada recuros implementado en esa web.

Esta es una consulta de Google.com

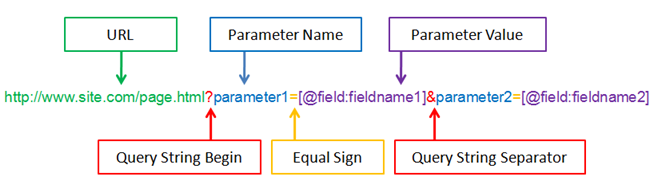


## ESTRUCTURA DE UNA URL:

PARAMETROS QUERYS:

SIRVEN PARA FILTAR BUSQUEDAS SE COMPONEN DE PARAMETROS CLAVE VALOR.



# Modulo url:

Supongamos una URL

*const* miURl = new URL

('https://www.yisusprograming.god/cursos/programacion?backend=python&frontend=javascript');

Gracias a el módulo url nosotros podríamos extraer data de las diferentes partes de la misma una vez que esta llega al servidor.

##### Acceder a l nombre del host:

console.log(miURl.hostname);



##### Acceder al camino:

console.log(miURl.pathname);



##### Acceder a los parámetros querys:

console.log(miURl.searchParams);



##### Acceder a cada parámetro querys:

console.log(miURl.searchParams.get('backend'));

console.log(miURl.searchParams.get('frontend'));



# ROUTING:

PERMITE MANEJAR LAS SOLICITUDES DE LOS CLIENTES DEPENDIENDO DE CIERTOS CRITERIOS.

EL PRIMER CRITERIO ES EL METODO, DE ESTA FORMA EL SERVIDOR SABE COMO EMPEZAR ESTA SOLICITUD.

EL SEGUNDO CRITERIO ES EL PATH. OSEA HACIA DONDE VA DIRIGIDO ESTE METODO.

BASICAMENTE RESUMIENDO EL ROUTING SE COMPONE DE:



QUE VA A SER

DONDE VA A SER

COMO LO VA A HACER

\*El código de estado por defecto que se asocia a las solicitudes es 200\*

Para los otros códigos de estado es necesario configurarlos manualmente.

NODEMON:

Nodemon permite refrescar node cuando detecta algún cambio en los archivos; de esta forma no es necesario reiniciar la terminal y volver a cargar el archivo.

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

*const* PUERTO = 3000;

*const* servidor = http.createServer((*req*,*res*)*=>*{

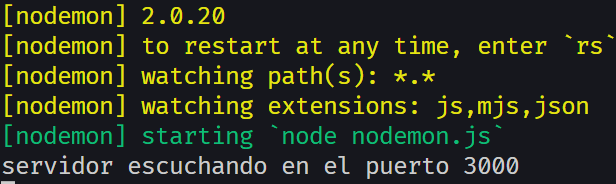
*res*.end('PROBANDO NODEMON')

})

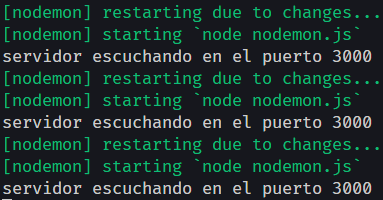
servidor.listen(3000,()*=>*{console.log(`servidor escuchando en el puerto ${PUERTO}`)});



Y automáticamente veriamos esto:



Ahora cada vez que guardemos el archivo pasara este mensaje.



\*aquí se apreto 3 veces "ctrl+s”\*