**Crear un servidor básico con Node.js**

var http = require('http');

http.createServer(function (request, respuesta) {

  respuesta.writeHead(200, {'Content-Type': 'text/plain'});

  respuesta.write("Hola Mundo!");

  respuesta.end();

}).listen(3000, 'localhost');

console.log('Server running at http://loccalhost:3000/');

### Instalar nodemon

Nodemon sirve para detener, reiniciar y mostrar los cambios el Sistema NodeJS: npm install nodemon -g

Luego para arrancar el sist, en vez de llamar node, llamamos nodemon: nodemon servidor.js

### Instalamos un servidor Nginx en Windows

En cmd escribirpara cambiar de directorio : cd + (la direccion de la carpeta de NGINIX)

Esto cambiara el directorio de cmd a nginix y para ejecutar nginx: start nginx O nginix

Probar que funcione en la direccion “localhost”, verás en msj de bienvenida.

### Instalamos un servidor Nginx en Windows

El mensaje de bienvenida de nginx está en el archive index.html en la carpeta html de nginx.

Para cambiar la ruta hacia los archivos que quieres que se muestre en localhost, debemos ir a la carpeta config de nginx a: nginx.conf

Hay un apartado de server, location(hay q asegurarse q no tenga #):

location / {

            root   html;

            index  index.html index.htm;

cambiar a:

location / {

            root   C:\Servidor;

            index  index.html index.htm;

“C:\Servidor” fué creada por mi y dentro hay un index con un mensaje cualquiera.

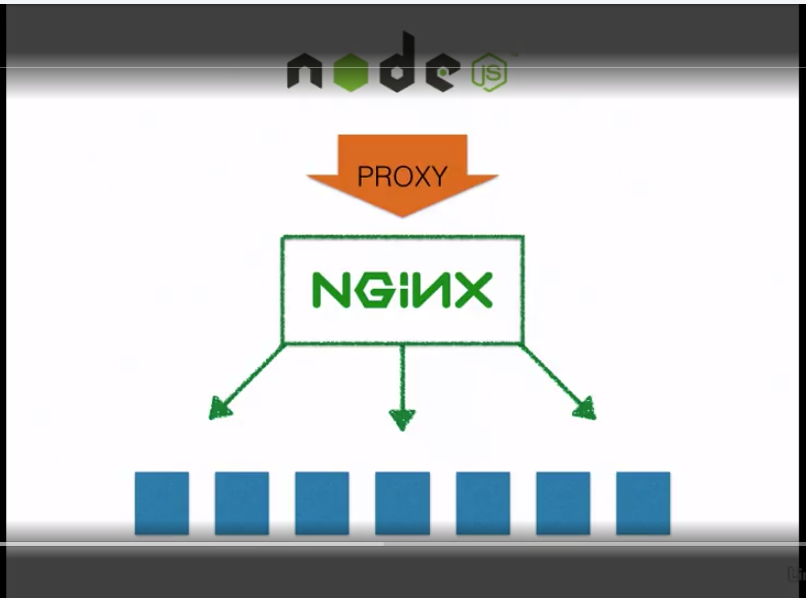
Luego, con la carpeta seleccionada de NginX en CMD, se inicializa nginx escribiendo: nginx

NOTA: para parar nginx demeos escribie en consola: nginx -s stop

Una vez inicializado, en la direccion localhost deberiamos ver el mensaje de index.html de la carpeta C:\Servidor

### Crear un sitio web básico con Nginx y Node.js

cómo conectar un servicio de Node.js con un servidor Nginx. Básicamente lo que vamos a hacer es que vamos a comenzar trabajando con un servicio de Node. Node se va a encargar de convertir los elementos de JavaScript que nosotros estamos trabajando en un documento que nos va a funcionar a nivel de servidor. Una vez que tengamos eso, vamos a crear lo que se llama una conexión por proxy. Esto va a ser básicamente una conexión indirecta, o un puenteo, que nos va a llevar directamente a Ngnix. Node.js va a conectarse directamente con Ngnix y Ngnix va a fungir aquí como un servidor, servidor que se va a encargar de distribuir toda la información a los diferentes servidores, equipos, clientes y demás elementos que ocupen nuestra información. Entonces el proceso de nuevo va a ser producir a través de Node un servicio y vamos a trabajar nuestro JavaScript utilizando Node. A partir de ahí, vamos a hacer un puenteo o un proxy que va a enviar la información. YNgnix va a consumir exclusivamente la información que nosotros estamos enviando desde Node. Y a partir de ahí se va a encargar de distribuirla para los diferentes elementos que la vayan a necesitar. Vamos a implementar esta arquitectura dentro de nuestro sistema.



El proxy:

En la carpeta conf de nginx, abrimos el archivo nginx.conf y en la parte Server => location pegar el codigo estandar del Proxy:

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IR $remote\_addr;

proxy\_set\_header X-Forwarded-For

$proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-NginX-Proxy true;

proxy\_pass http://localhost:3000/;

proxy\_redirect off;

Deberia verse asi:

    location / {

            root   C:\Servidor;

            index  index.html index.htm;

            proxy\_set\_header Host $host;

            proxy\_set\_header X-Real-IR $remote\_addr;

            proxy\_set\_header X-Forwarded-For

            $proxy\_add\_x\_forwarded\_for;

            proxy\_set\_header X-NginX-Proxy true;

            proxy\_pass http://localhost:3000/;

            proxy\_redirect off;

        }

Asi se conecta NodaJS con NginX. Amboes tienen que tener la misma direccion, en este caso: localhost:3000

Básicamente lo que está haciendo esto es que está diciendo que toda la información, todo el tráfico que llegue al servidor Ngnix, va a utilizar la información que viene de 'localhost', puerto 3000. En este caso, es la dirección que está dentro de mi computador, pero puede ser cualquier dirección IP o la dirección que tú necesites. Vamos a necesitar entonces que Node esté activo y que Ngnix esté activo. Solo falta activar los diferentes servidores para poder mostrar la información.

Hay que activar ambos servidores.

Comenzamos por activar NginX desde CMD.

Luego activamos NodeJS desde CMD o la terminal de Visual Studio Code con Nodemon: nodemon servidor.js

### Rutas básicas en Node.js

Gestionar rutas dentro de Node.js utilizando únicamente los elementos básicos de este sistema. Estoy utilizando el método 'createServer', lo cual va a crear un servidor, y lo estoy trabajando específicamente en el puerto 3000 dentro de mi 'localhost'. Esto es básicamente un servidor. Sin embargo, mi servidor aún no tiene ninguna respuesta. Dentro de mi servidor, yo tengo dos parámetros. Tengo uno que se llama 'petición', el primero, y el segundo parámetro en este caso lo he llamado 'respuesta'. El primer parámetro que recibimos va a tener información de lo que solicita el usuario de mi servidor. Por ejemplo, si el usuario quiere entrar a un documento llamado 'preguntas.html', dentro de 'petición' va a venir la información correspondiente a la ruta que está solicitando el usuario, además de que va a incluir información del usuario y mucha otra información relacionada con la petición. También voy a tener un parámetro llamado 'respuesta', y en este caso a ser el resultado de la información que yo le voy a devolver al usuario. Primero, voy a definir cuál va a ser el tipo de dato que voy a devolver. Voy a utilizar 'respuesta.writeHead'.

Ahora, para saber cuál es la información que me está pidiendo el usuario, yo tengo acceso a 'petición'. Dentro de 'petición' está toda la información que el usuario está pidiendo, en este caso, la ruta, así que yo tengo acceso a un valor dentro de 'petición' que se llama 'url'. 'Url' tiene específicamente el enlace que me está solicitando el usuario. Yo lo que voy a hacer es que voy a generar acá una estructura lógica llama 'switch', que me ha a permitir a mí devolver diferentes respuestas dependiendo de la información que me solicitó el usuario. Así que voy a comenzar haciendo este 'switch'.

var http =  require ("http");

var fs = require('fs');

http.createServer(function(peticion,respuesta){

    respuesta.writeHead(200, {'Content-Type': 'text/html'});

    switch (peticion.url) {

        case '/':

            plantilla = "inicio.html";

            break;

        case '/nodejs':

            plantilla = "sobrenode.html";

            break;

        default :

            plantilla = "404.html";

            break;

    }

    fs.readFile( "./plantillas/" + plantilla , function(error, datos) {

        console.log("error",error, plantilla, datos.toString() );

        respuesta.write(datos.toString());

        respuesta.end();

    });

}).listen(3000, 'localhost');

console.log("Server funcinando");

Ahora, para poder acceder a los archivos del sistema, vamos a necesitar invocar una librería de Node que se llama 'file system'. en este caso voy a crear una variable que se llama 'fs' por "file system" y voy a invocar la librería 'file system'. Esto me va a dar a mí acceso a los archivos del sistema y poder invocar y administrar estos documentos en HTML.

Así que vamos a decirle: "Quiero que lea un archivo específico". ¿Cuál archivo? Bueno, primero quiero que busque dentro de el "folder", o carpeta, llamada 'plantillas –que está acá– /' y le voy a decir que utilice el nombre de la variable que estamos usando acá. Entonces dependiendo de la ruta a la que haya entrado, va a buscar dentro de 'plantillas' un diferente archivo.

### Instalamos Express

Aprenderemos ahora cómo instalar y configurar Express para gestionar de una manera más eficiente y sencilla las rutas dentro de Node.js.

antes de comenzar a utilizar Express, necesito instalarlo. Express lo debemos instalar en cada proyecto. “npm install express”

en app.js para usar express debemos crear “var express = require(‘express’)” todos los elementos que están descargados dentro de esta librería están disponibles dentro de mi aplicación.

var express = require('express');

var app = express();

app.get('/' , function( peticion, respuesta){

    respuesta.send('Hola mundo con express')

 })

 app.listen( 3000 , function(){

     console.log("escuchando en el puerto 3000");

 })

activar mi aplicación. Vamos a poner 'app.listen'. Y vamos a decir cómo va a trabajar mi aplicación. Vamos a decirle que trabaje en el puerto 3000 y vamos a darle una función acá, que va a activarse cada vez que nosotros hagamos una petición dentro de este puerto. Básicamente ahora lo que voy a poner es 'console.log' y le vamos a poner 'escuchando en el puerto 3000'. Ya tenemos nuestra aplicación lista, ahora nada más nos falta ejecutarla. Vamos a la consola. Esta información que nos está apareciendo dentro de la consola es toda la información que acabamos de descargar de Node. Y ahora voy a poner 'nodemon app.js' para activar mi aplicación.

### Configuramos la ruta base en Express

var express = require('express');

var fs = require('fs');

var app = express();

app.use (express.static(\_\_dirname + '/public') );

app.get('/' , function( peticion, respuesta){

    respuesta.sendFile('index.html')

 })

 app.listen( 3000 , function(){

     console.log("escuchando en el puerto 3000");

 })

Voy a definir una ruta, voy a poner 'app.get', defino la raíz del documento, defino una función que se activará en el momento en que el usuario solicite esta ruta específica, voy a definir la petición y la respuesta. Siempre han de venir en este orden estos dos parámetros.

Una vez que tengo definida la instancia de Express, voy a utilizar un comando llamado 'app' –en este caso el nombre de la instancia–, y el comando se va a llamar 'use'. en este caso lo que le estoy diciendo es que voy a utilizar una ruta específica. Le voy a decir que defina una ruta específica dentro Express. Así que le vamos a poner 'express.static' para llamar a todos nuestros archivos estáticos. Y esto vamos a definirlo con una variable. En este caso, vamos a tomar un par de valores variables. Primero 'dirname', que lo que va a hacer es que va a tomar el nombre del directorio en el que estamos trabajando y, dentro de este directorio, este caso 'express\_sitioweb', vamos a buscar la carpeta 'public', que es la que yo tengo acá. Así que básicamente lo que estoy diciendo al sistema es que a partir de ahora, cada vez que busque un archivo y no esté muy seguro de dónde se encuentra como, por ejemplo, en este caso 'index', que solo estoy poniendo el nombre del archivo, entonces lo va a buscar directamente dentro de la raíz de la carpeta 'public'. Como vemos, 'index' sí se encuentra dentro de la carpeta 'public'.

Si escribo acá 'tienda. html', a pesar de que no está definido dentro de las rutas de Express, voy a poder entrar a la dirección que ya definí porque inmediatamente el sistema va a buscarlo dentro de la carpeta 'public'. Y dice: "Bueno, dentro de la carpeta 'public' hay un archivo, 'tienda.html'", y por eso no lo a mostrar. Así que nosotros con esta ruta base podemos básicamente crear un archivo, por lo menos un sitio completo de manera estática, utilizando esta ruta. Y así nos vamos a asegurar de que todos los archivos que estamos sirviendo, en el caso de que no estén definidos, van a buscarse automáticamente dentro de esta ruta.

### Instalación de Pug para Node.js

En esta lección vamos a aprender cómo instalar Pug. Pug es un sistema que nos va a permitir desplegar información dinámica a través de plantillas.

“npm install pug”

Dentro de los archivos base de mi aplicación, tengo una carpeta llamada 'views' y dentro de esta carpeta tengo un archivo llamado 'index.pug'. La carpeta 'views' ya viene predefinida dentro de la configuración original de Pug.

dentro de Express, acá dentro de la configuración de Express, justamente en la sección que define la ruta y la información que va a desplegar mi aplicación al momento de iniciar, yo voy a llamar acá este valor, que va a ser 'respuesta', y voy a llamar directamente el comando 'render'. Dentro de 'render' voy a definir el nombre de mi plantilla, que en este caso va a ser 'index.pug'.

Básicamente le decimos que utilizamos como base la carpeta 'public', donde tenemos acá todos los archivos fuente de la aplicación. Pero le decimos que, en el caso de que la aplicación entre a la raíz, desplegamos los contenidos de esta plantilla llamada 'index.pug'.

var express = require('express');

var app = express();

const pug=require('pug');

app.use (express.static(\_\_dirname + '/public') );

app.get('/' , function( peticion, respuesta){

     respuesta.render('index.pug')

})

 app.listen( 3000 , function(){

     console.log("escuchando en el puerto 3000");

 })

Así que, en el caso de que tuvieras algún problema para desplegar la información de tu plantilla, revisa que no exista un archivo con el mismo nombre dentro de la carpeta 'public'.

### Sintaxis de Pug

Normalmente en HTML tenemos elementos como esto, llamados etiquetas. En el caso de Pug, no vamos a utilizar etiquetas. Vamos a utilizar un sistema de indentación y vamos a llamar directamente el nombre de la etiqueta, pero sin utilizar ningún tipo de carácter adicional.

en Pug simplemente llamo la etiqueta 'head' y dentro de ella voy a utilizar la tecla Tab. Como puedes ver acá, ahora hay una indentación y esta indentación va a definir que todos los elementos que aparezcan acá indentados van a estar dentro de la etiqueta 'head'.

Idex.pug

doctype html

html(lang="es")

head

    title este es el nombre de mi app

body

    h1

        a(href="ejemplo\_pug.org") Bienvenido a mi tienda

    div

        p Nuestra tienda de camisetas ofrece lo mejor, satisfacción 100% garantizada

        img(src="images/camiseta1.png", alt="camiseta")

    h2 Nuestras Camisetas

    ul

        li camiseta 1

        li camiseta 2

        li camiseta 3

        li camiseta 4

### Datos dinámicos en Pug

Si te fijas, estoy utilizando la sintaxis de objeto, así que estoy poniendo llaves. Luego pongo el nombre de la propiedad. En este caso, 'titulo:' y entre comillas pongo el valor que va a tener. Luego pongo una coma y repito el proceso.

En app.js

app.get('/' , function( peticion, respuesta){

     respuesta.render('index.pug',{

         titulo:"Nombre de mi app.",

         textoParrafo:"Nuestra tienda de camisetas ofrece lo mejor, satisfacción 100% garantizada"

     })

})

En index.pug

head

    title #{titulo}

body

    h1

        a(href="ejemplo\_pug.org") Bienvenido a mi tienda

    div

        p #{textoParrafo}

        img(src="images/camiseta1.png", alt="camiseta")

tenemos acá nuestra plantilla de Pug utilizando valores dinámicos, simplemente con una pequeña plantilla, con un pequeño formato, que es, consiste en utilizar el signo numeral, llaves y el nombre de la variable. Esta es la forma en que podemos desplegar rápidamente valores dinámicos dentro de nuestras plantillas.

### Arreglos en Pug

En esta lección aprenderemos a iterar generando bucles que desplieguen información dentro de nuestras plantillas de Pug.

Lo primero que voy a hacer es insertarle una hoja de estilos dirigiéndome específicamente al apartado donde dice 'head'. Acá en este caso dice 'title' y dentro de esta sección vamos a poner un apartado llamado 'link'. Esto va a generar la etiqueta 'link', que es la que nos va a relacionar con una hoja de estilos. También le vamos a poner la propiedad 'rel', 'stylesheet' y ponemos una última opción que es 'href', y vamos a apuntar a una hoja de estilos. Acá, dentro de 'public', yo ya tengo una hoja de estilos que se va a llamar 'estilos.css' y está dentro de la carpeta 'CSS', así que vamos a poner '/css/ estilos.css'.

En app.js

camisetasLista = [

    {color:"Naranja" ,  imagen:"images/camiseta1.png"},

    {color:"Roja" ,  imagen:"images/camiseta2.png"},

    {color:"Azul" ,  imagen:"images/camiseta3.png"},

    {color:"Verde" ,  imagen:"images/camiseta4.png"}

]

En tienda.pug

doctype html

html(lang="es")

head

    title Tienda

    link(rel="stylesheet", href="/css/estilos.css")

body

    h1

        a(href="tienda\_de\_fruta.com") Tienda de Camisetas

    each camiseta in camisetas

        div.item

            h2 Camiseta #{camiseta.color}

            img(src=camiseta.imagen , alt="camiseta")

ya se muestra un valor por cada uno de los ítems que se encuentran en este arreglo. Esta es la forma en que podemos mostrar muy rápidamente y sin mayor problema información que se encuentra dentro de un "array" dentro de nuestras plantillas de Pug.

### Crear rutas avanzadas de Node.js

Las rutas estáticas, básicamente, nos presentan una ruta que no va a cambiar. Esta ruta siempre va a ser exactamente igual y no va a pasar ningún cambio cada vez que entremos a ella. Vamos a hacer una ruta dinámica que va a cambiar de acuerdo a la información que recibimos desde el enlace.

“/:color” es un parametron dianmico

App.js

app.get('/tienda/comprar/:color' , function( peticion, respuesta){

    let datosDeCamiseta = camisetasLista.filter( function(item){

        if( peticion.params.color == item.color){

            return item

        }

    })[0]

    respuesta.render('detalles.pug' , {

        color: peticion.params.color,

        datos: datosDeCamiseta

    } );

})

“peticion.params” aqui params toma como parametro color de “/:color”

Se crea una nueva propiedad, en este caso una variable que va a ser 'datos De Camiseta'. 'datosDeCamiseta' va a tomar la información de 'camisetasLista' que, en este caso, es el arreglo que yo tengo donde se encuentra toda la información, y vamos a aplicar acá un 'filter'. El filter separa cada elemento del arreglo, luego se ejecuta una funcion para cada elemento filtrado y si un elemento de peticion.params.color es exactamente igual a algun elemento de arreglo como item (item.color) , devuelve el elemento del arreglo (item) osea el color y la imagen.

En datosDeCamiseta se guarda un item (elemento del arreglo)

Luego amtcheamos datosDeCamiseta en “datos”

Detalles.pug

El #{color} viene de el parametron dinamico :/color de app.js

En “datos” entraria todo los elementos de datosDeCamiseta (color e imagen) pero aqui solo pediriamos por la iamgen

body

    h1

        a(href="ejemplo\_pug.org") Comprar camiseta color #{color}

    div

        h2 Vista previa

        img(src=datos.imagen, alt="camiseta")

en tienda.pug

Ahora voy a incluir acá también un enlace. Este enlace va a generar una ruta que va a ser 'tienda/comprar' y muestra el valor del color. Así vamos a activar los valores que tenemos dentro de nuestra ruta dinámica. Guardemos los cambios y veamos el resultado en nuestra aplicación. Tenemos la tienda, tenemos el bucle que está desplegando la información. cada uno de estos elementos muestra los ítems que tenemos en un arreglo y, al hacer clic acá, en este botón, me va a enviar a la ruta dinámica enviando el parámetro del color. Al obtener eso, mostramos esta información pero también podemos enlazarla con la información que se encuentra en el arreglo y obtener más datos como, en este caso, la ruta a la imagen.

each camiseta in camisetas

        div.item

            h2 Camiseta #{camiseta.color}

            img(src=camiseta.imagen , alt="camiseta")

            a(href="tienda/comprar/"+camiseta.color) Comprar

### Crear páginas de error con Node.js

En app.js

app.use ( function(peticion, respuesta){

    respuesta.status(400);

    let URLerror = peticion.originalUrl;

    respuesta.render('404.pug' , {textoError: URLerror});

})

En 404.pug

doctype html

html(lang="es")

head

    title #{titulo}

    link(rel="stylesheet", href="/css/estilos.css")

body

    h1

        a(href="/") Bienvenido a mi tienda

    h2  No encontramos "#{textoError}" en el sistema.

respuesta.status = estado de la pagina (200 la pagina funciona correctamente y 400 incorrectamente/error)

respuesta.render = para desplegar una plantilla

### Instalación de Express Generator

Crear una aplicación exprés puede ser muy complicado. Tenemos que crear una serie de archivos, utilizar una estructura específica y, además, configurar una serie de diferentes detalles. Precisamente para solucionar ese problema existe un paquete llamado Express Generator. Este nos va a permitir crear una aplicación en solo unas cuantas líneas de código.

Perimero verificar en CMD que temenos Node como NPM: “node -v” y “npm -v”

Node package manager = npm

Luego, instalamos el generador: “npm install express-generator -g”

Revisar la version: “express --version”

### Proyectos Express con NPM

En esta lección te voy a mostrar cómo crear desde cero una aplicación con Express y Node utilizando el Express Generator.

1º en cmd escribir: “cd + el path de una carpeta vacia (prueba)” (prueba es un nombre cualquiera)

2º instalar pug: “express --view =pug prueba”

Esto genera el Proyecto y ahora hay q seguir las instrucciones q aparecen en consola:

1. cd prueba
2. npm install (esto instala dependencias de npm, se crea node\_modules)
3. SET DEBUG=prueba:\* & npm start (define un estado de depuración y también inicializa el servidor)

Si todo sale bien, nos daria una direccion localhost para darnos la bienvenida (ej localhost:3000)

### Instalación y configuración de MongoDB y Robomongo en Windows

Dentro de un sitio basado en Node.js podemos utilizar prácticamente cualquier tipo de base de datos. Sin embargo, posiblemente la base de datos más popular entre los desarrolladores de Node.js es MongoDB. Esta es una base de datos no relacional bastante eficiente y de código abierto. Es muy, muy popular y puedes descargarla desde 'mongodb.org'.

Luego en C:/ ahy que crear una carpeta llamada “data” y dentro de data “db”, estos son los requisites de MongoDB. Para ejecutar Mongo hay que ejecutar un archive llamado “mongod.exe” dentro de la carpeta “bin”

podemos administrar Mongo desde una interfaz gráfica. Yo te recomiendo que utilices este programa llamado Robomongo, que lo puedes descargar desde 'robomongo.org'. Es un programa gratuito y lo puedes descargar para cualquier versión de sistema operativo que estés utilizando.

Crear una conexion:

Robo 3T(Robomondo ) > Create >localhost > Save y doble click en la nueva conexion

Importante: siempre tienes que tener abierta esta conexión de MongoDB para estar seguro de que está funcionando.

Importante: siempre tienes que tener abierta esta conexión de MongoDB para estar seguro de que está funcionando.

### Obtener información de la base de datos para Node.js

Vamos a crear ahora un proyecto basado en Express y Node.js que utiliza MongoDB para obtener los datos de la aplicación.