UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

CURSO:

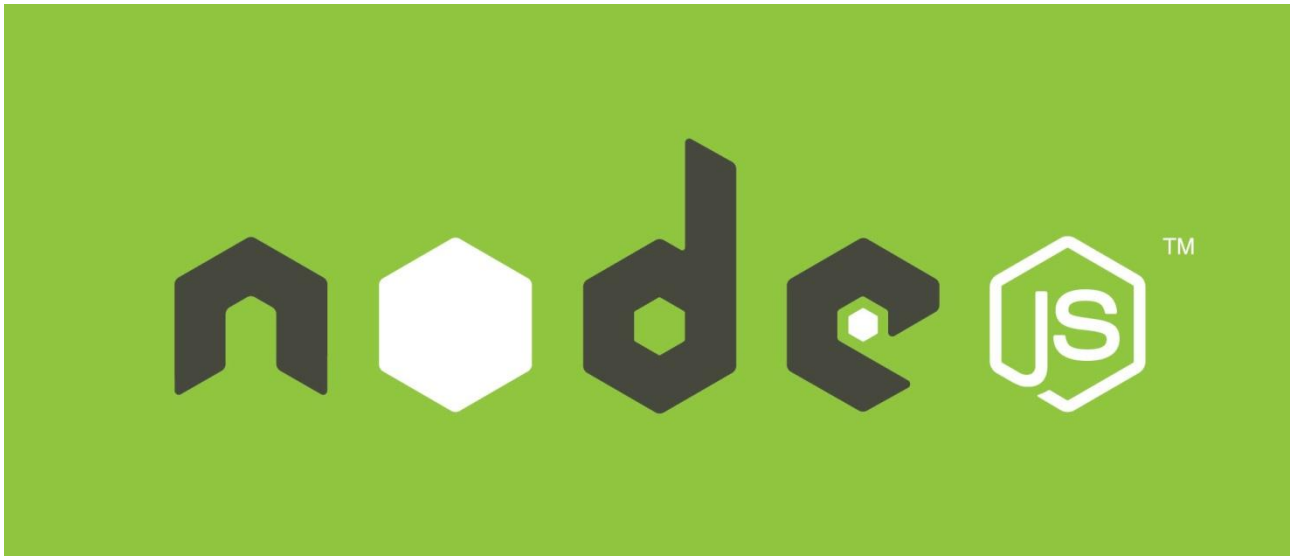
Desarrollo en Node JS

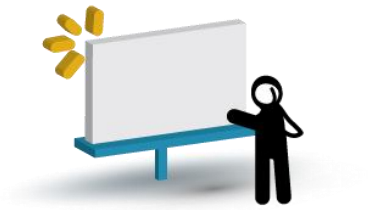
Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

UTN.BA UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL BUENOS AIRES | Centro de e-Learning

Unidad 2:

NodeJSPackages – Eventos y Streams





Presentación:

En esta segunda Unidad del curso, continuamos viendo los conceptos fundamentales de Node JS, vamos a ver qué es el gestor de paquetes de Node, npm, para qué sirve y cómo lo utilizamos.

Continuamos con ejemplos prácticos del uso del módulo http, vemos cómo servir archivos estáticos html, css, jpg, mp3, mp4, etc. También cómo recuperar datos de un formulario y cómo recuperar datos de los parámetros por GET.

Objetivo:

Al terminar la Unidad los participantes:

- Sabrán cómo recibir datos de un formulario y cómo recibir datos enviados en la url por GET
- Conocerán el gestor de paquetes npm.
- Sabrán cómo instalar paquetes de manera global.

Bloques temáticos:

1. HTTP
    - Datos que envía el navegador
    - Servidor web que sirve paginas html
    - Servir archivos estáticos css, jpg, etc.
    - Crear una cache
    - Recuperar datos de un formulario
    - Libro de visitas mediante un archivo de texto
    - Recuperar datos de una url (GET)
2. NPM
    - Comando npm para gestión de paquetes
    - Instalar paquetes de manera global con npm
    - Ejemplo con Nodemailer
    - Ejemplo de upload de archivos con modulo formidable
    - Comunicación con Mysql utilizando el módulo mysql
    - Nodemon

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

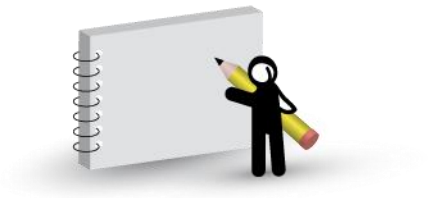
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

- *El MEC es el modelo de E-learning colaborativo de nuestro Centro.*



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1 – http

En la unidad anterior vimos cómo crear un servidor web a través del módulo http de NodeJS, veremos a continuación como interpreta el pedido del navegador el servidor web que iniciamos con el modulo http, como servir archivos estáticos html, archivos css, jpg, mp3, mp4, etc y como tomar datos de un formulario o de una url con GET.

También veremos el gestor de paquetes npm de NodeJS del que ya hemos mencionado anteriormente y como lo usamos para incluir paquetes.

Datos que envía el navegador

En la unidad anterior vimos cómo crear un servidor web a través del módulo http de NodeJS, para ello creábamos una función asíncrona que capturara las peticiones que envía el navegador a través del siguiente código:

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

var servidor=http.createServer(function(pedido,respuesta){
   respuesta.writeHead(200, {'Content-Type': 'text/html'});
   respuesta.write('<!doctype html><html><head></head>'+
                '<body><h1>Sitio en desarrollo</h1></body></html>');
   respuesta.end();
});

servidor.listen(8888);

console.log('Servidor web iniciado');
```

Cada vez que sucede una petición de recurso (puede ser una página, una imagen, un archivo de sonido etc.) se ejecuta la función anónima que le pasamos a createServer.

Vamos a hablar ahora del primer parámetro que recibe esta función, pedido.

Este objeto tiene muchos datos que llegan al servidor. Analizaremos la propiedad url del mismo.

Se trata de un string con la ubicación exacta del recurso que pide el navegador al servidor, por ejemplo:

http://localhost:8888/carpeta1/pagina1.html?parametro1=10parametro2=20

Estamos solicitando al servidor localhost el archvio pagina1.html que se encuentra en la carpeta 'carpeta1' del servidor y tiene dos parámetros.

Si codificamos el ejercicio9.js con el siguiente código:

```
var http=require('http');
var url=require('url');

var servidor=http.createServer(function(pedido,respuesta){
   var objetourl = url.parse(pedido.url);
   console.log('camino completo del recurso y parametros:'+objetourl.path);
   console.log('solo el camino y nombre del recurso     :'+objetourl.pathname)
   console.log('parametros del recurso                  :'+objetourl.query)
   respuesta.writeHead(200, {'Content-Type': 'text/html'});
   respuesta.write('<!doctype html><html><head></head><body>Hola mundo</body></html>');
   respuesta.end();
});

servidor.listen(8888);

console.log('Servidor web iniciado');
```

Lo ejecutamos en la consola y luego vamos al navegador y tipeamos la ruta antes mencionada:

http://localhost:8888/carpeta1/pagina1.html?parametro1=10parametro2=20

Aparecerán en la consola los datos correspondientes a la ruta, el recurso y los parámetros del mismo.

No es necesario que estén creados la carpeta carpeta1 y el archivo pagina1.html para probarlo.

Analicemos un poco el código:

Requerimos el módulo llamado 'url' que nos facilita el análisis de las distintas partes de una url además del módulo 'http'.

```
var http=require('http');
var url=require('url');
```

En la función anónima llamamos al método parse del objeto 'url' y le pasamos como parámetro la propiedad url del objeto pedido que llega a la función:

```
var objetourl = url.parse(pedido.url);
```

Esta actividad lo que hace es parsear (procesar el string que contiene pedido.url) y descomponernos las distintas partes de una url en un objeto literal para que sea más fácil analizar su contenido.

El objeto literal objetourl tiene tres propiedades fundamentales para recuperar los datos en forma parcial y agrupada de la petición HTTP que hizo el navegador:

```
console.log('camino completo del recurso y parametros:'+objetourl.path);
console.log('solo el camino y nombre del recurso     :'+objetourl.pathname)
console.log('parametros del recurso                  :'+objetourl.query)
```

Con estos datos mediante if podemos redirigir el pedido al recurso respectivo (por el momento siempre nuestro sitio devuelve el 'Hola mundo').

Hay otra cosa que debemos nombrar: los navegadores la primera vez que acceden a un sitio piden un recurso (el archivo favicon.ico) y es eso por lo que se ejecuta dos veces la función anónima, una para solicitar el archivo 'archivo1' y otra para solicitar el archivo 'favicon.ico' ya veremos cómo retornar dicho archivo y otros que se soliciten.

Servidor web que sirve páginas estáticas html

Vamos a ver como dada una petición de una página HTML proceder a verificar si existe dicha página, leerla y retornarla al navegador que la solicitó.

Vamos a crear un subdirectorio llamado 'static' (puede tener cualquier nombre) en la carpeta donde crearemos nuestra aplicación Node.js y almacenaremos tres páginas HTML: index.html, pagina1.html y pagina2.html

Por otro lado creamos el archivo js que levanta el servidor web y que devuelve las paginas solicitadas según el pedido del navegador.

El código del archivo ejercicio10.js es el siguiente:

```
var http=require('http');
var url=require('url');
var fs=require('fs');

var servidor=http.createServer(function(pedido,respuesta){
   var objetourl = url.parse(pedido.url);
   var camino='static'+objetourl.pathname;
   if (camino=='static/')
      camino='static/index.html';
   fs.exists(camino,function(existe){
      if (existe) {
         fs.readFile(camino,function(error,contenido){
            if (error) {
               respuesta.writeHead(500, {'Content-Type': 'text/plain'});
               respuesta.write('Error interno');
               respuesta.end();
            } else {
               respuesta.writeHead(200, {'Content-Type': 'text/html'});
               respuesta.write(contenido);
               respuesta.end();
            }
         });
      } else {
         respuesta.writeHead(404, {'Content-Type': 'text/html'});
         respuesta.write('<!doctype html><html><head></head><body>Recurso inexistente</body></html>');
         respuesta.end();
      }
   });
```

```
});

servidor.listen(8888);

console.log('Servidor web iniciado');
```

Para probar el ejemplo tenemos que ejecutar desde la consola ejercicio10.js e ir al navegador y tipear localhost:8888.

Tienen comentado el código del ejercicio10.js para que puedan seguirlo y entender el funcionamiento.

Servir archivos estáticos css, jpg, etc

En el ejemplo anterior vimos cómo crear un servidor web que sirva solo paginas html, pero sabemos que en la realidad con esto no alcanza, un sitio web está constituido por otro tipo de archivos como css, jpg, gif, mp3, mp4, entre muchos otros.

Siempre que devolvemos un recurso a un cliente (navegador) en la cabecera de respuesta indicamos el tipo de recurso devuelto:

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

En el Content-Type indicamos el tipo de recurso a devolver, dependiendo de qué tipo de archivo se trate tenemos que indicar un Content-Type diferente.

Para eso se utiliza el MIME type (*Multipurpose Internet Mail Extensions*). El IANA (Internet Assigned Numbers Authority) que es un departamento de ICANN es el organismo oficial encargado de estandarizar y publicar estas clasificaciones.

Se puede ver una lista completa de los MIME types en el sitio: ***http://www.sitepoint.com/web-foundations/mime-types-complete-list/***.

Algunos ejemplos son:

Content-type: text/html
Content-type: text/css
Content-type: image/jpg
Content-type: image/x-icon
Content-type: audio/mpeg3
Content-type: video/mp4

Content-type: application/json
Content-type: application/x-www-form-urlencoded
Content-type: multipart/form-data

Para probar esto creamos ejercicio11.js y basándonos en el ejemplo anterior y haciendo algunas modificaciones al código para que pueda servir otros tipos de archivo y no solamente paginas html.

Para eso creamos un array con algunos MIME types:

```
var mime = {
   'html' : 'text/html',
   'css'  : 'text/css',
   'jpg'  : 'image/jpg',
   'ico'  : 'image/x-icon',
   'mp3'  : 'audio/mpeg3',
   'mp4'  : 'video/mp4'
};
```

El resto del código es igual al ejemplo anterior salvo el momento en donde devolvemos el archivo:

```
var vec = camino.split('.');
var extension=vec[vec.length-1];
var mimearchivo=mime[extension];
respuesta.writeHead(200, {'Content-Type': mimearchivo});
respuesta.write(contenido);
respuesta.end();
```

Que obtenemos del archivo la extensión y buscamos en el array el Content-Type que tenemos que devolver.

En nuestro ejemplo hemos limitado a servir 6 formatos de archivos distintos. Para servir otros formatos de archivos deberíamos agregar al objeto literal mime los valores respectivos que define el protocolo MIME.

Crear una cache

En el ejemplo anterior vimos cómo crear un servidor web con NodeJS que nos devuelve distintos formatos de archivos.

La mecánica que empleamos es que en casa solicitud del cliente (navegador) analizamos primero si dicho recurso se encuentra en el servidor y en caso afirmativo procedemos a leerlo mediante las funciones que provee el módulo 'fs'. Finalmente lo retornamos al navegador que solicito el recurso indicando del tipo de archivo que se trata para que el navegador pueda proceder a mostrarlo si es una imagen, renderizar si es un HTML etc.

Vamos a hacer una modificación al programa que hemos planteado hasta el momento para reducir drásticamente los tiempos requeridos de acceso al disco duro del servidor donde se almacenan los archivos, teniendo en cuenta que el acceso a disco por operaciones de I/O son costosas en tiempo.

Lo que vamos a implementar es un sistema de cache en el servidor para almacenar el contenido de los archivos estáticos (es decir aquellos que no cambian) y tenerlos almacenados en la memoria RAM.

Para eso modificamos un poco el código del ejercicio anterior, el programa hará lo mismo, arranca un servidor web de página estática y almacena las páginas pedidas en una cache para no tener que leerlas nuevamente del disco.

Para eso definimos un objeto vacío donde almacenaremos los nombres de los recursos y los contenidos de los mismos

```
var cache={};
```

Cada vez que es pedido un recurso al servidor corroboramos si el contenido está almacenado en la cache o no. La primera vez que es solicitado el contenido no estará en la cache por lo tanto la leerá del disco y la almacenará en la cache para las siguientes solicitudes.

Entonces ante un pedido de recurso verificamos la existencia del recurso en la cache, si esta se devuelve de allí, sino se devuelve del disco y se almacena en la cache para los sucesivos pedidos.

El código de este ejercicio lo encuentran en la carpeta ejercicio12.

Hay que tener en cuenta que estamos hablando de una cache en el servidor y no la cache del navegador. Es decir que cuando un cliente1 accede al sitio web y solicita el archivo: index.html a partir de este momento dicho archivo queda almacenado en la cache y cuando un cliente2 acceda al sitio y solicite el archivo: index.html su contenido lo recuperará de la cache y no del disco (esto hace mucho más eficiente a nuestro programa)

Cuando detenemos nuestro programa Node.js la cache es una variable de memoria por lo que su contenido se pierde. Al arrancar nuevamente el servidor a medida que pidamos recursos la cache si irá cargando.

Recuperar datos de un formulario

Hasta este momento hemos visto cómo podemos implementar un servidor de un sitio web que responde a peticiones de páginas y recursos estáticos que se encuentran en dicho servidor.

Ahora veremos cómo podemos hacer un programa en Node.js que pueda procesar los datos de un formulario HTML.

El código del ejercicio13 es una modificación del código que veníamos utilizando en los ejemplos anteriores, la principal diferencia es que hemos subdividido las actividades en distintas funciones para hacer más legible el programa, en lugar de trabajarlo con funciones anónimas como veníamos haciéndolo.

Van a encontrar comentado el código del ejemplo para que puedan comprender bien el funcionamiento. Y lo pueden probar tipeando node ejercicio13.js desde la consola y luego tipeando en el navegador localhost:8888.

Libro de visitas mediante un archivo de texto

Tomando el ejemplo anterior que captura los datos de un formulario html vamos a desarrollar un libro de visitas que tome los datos de un formulario y lo guarde en un archivo de texto.

El ejercicio14, que encontrarán en la carpeta con ese mismo nombre, muestra un index con la posibilidad de:

- Cargar comentarios o
- Mostrar todos los comentarios.

Al presionar Cargar comentarios se muestra un formulario y de la misma manera que en el ejemplo anterior procedemos a mostrar el formulario y una vez que el usuario ingresa los mismos y presiona el botón de submit se ejecutara la función grabarComentarios tomando los datos del formulario, de la misma forma en la que trabajamos en el ejemplo anterior y luego grabara los datos en el archivo.

En la función grabarEnArchivo concatenamos los datos a grabar y llamamos al método appendFile, otro de los métodos disponibles del módulo fs, que crea el archivo de texto 'visitas.txt' o lo abre para agregar en el caso que ya exista. El segundo parámetro que recibe es el string a grabar en el archivo de texto.

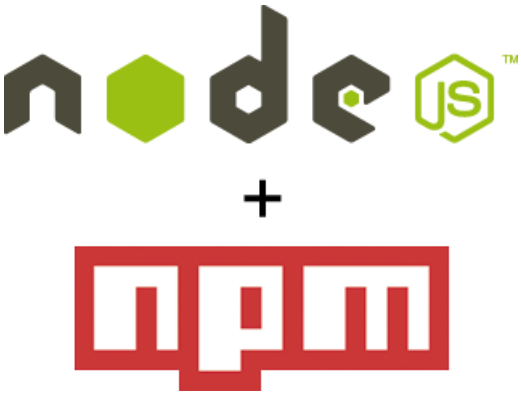
Por otro lado si presionamos desde el index la opción para leer los comentarios se ejecutara la función leerComentarios utiliza el objeto 'fs' para leer el archivo de texto y generar una página dinámica con dichos datos.

Recuperar datos de una url (GET)

Para explicar el paso de parámetros por la url y como tomarlos desde nuestro archivo js vamos a desarrollar un ejemplo en el cual en una primer pantalla se verán una lista de números del 1 al 20 al presionar en alguno de ellos deberá mostrar la tabla de multiplicar del numero presionado. El número presionado tiene que pasar como parámetro a través de la url.

El ejercicio lo encuentran en la carpeta ejercicio15 el código en su mayor parte es similar a los ejercicios que venimos viendo, solo tienen que prestar atención a la función listarTablaMultiplicar cuando accede al parámetro recibido por la url. El código esta comentado.

2 – NPM



Es una herramienta que ya instala Node.js y la podemos utilizar desde la línea de comandos. Nos permite reusar código de otros desarrolladores y también nos permite compartir nuestro código.

Comando npm para gestión de paquetes

Node.js viene por defecto con un conjunto limitado de módulos de uso general.

Nosotros podemos implementar toda la lógica para un determinado problema o buscar si otro desarrollador ya se le presentó dicho problema y compartió su código.

El sitio web ***npmjs*** es el repositorio oficial donde podemos compartir nuestros módulos y paquetes, y beneficiarnos de los módulos y paquetes que desarrollaron otros programadores.

Para darnos una idea de la cantidad de desarrolladores que están utilizando esta tecnología podemos decir que hay más de 180000 paquetes publicados en el sitio a la fecha.

Existen paquetes con muy diferentes objetivos como por ejemplos frameworks para sitios web (uno de esos llamado express), para acceder a una base de datos mysql o la base de datos mongoDB etc.

Existen distintos módulos que están disponibles de manera predeterminada en cualquier proyecto NodeJS y que por tanto no necesitamos traernos previamente a local mediante el gestor de paquetes npm. Esos toman el nombre de "Módulos nativos" y ejemplos de ellos tenemos el propio "http", "fs" para el acceso al sistema de archivos, "net" que es un módulo para conexiones de red todavía de más bajo nivel que "http" (de hecho "http" está mayormente escrito sobre el módulo "net" ***github.com/joyent/node/blob/master/lib/http.js***), "url" que permite realizar operaciones sobre "url", el módulo "util" que es un conjunto de utilidades, "child_process" que te da herramientas para ejecutar sobre el sistema, "domain" que te permite manejar errores, etc.

Hemos estado viendo ejemplos del uso de algunos de estos módulos que mencionamos y continuaremos viendo durante el resto del curso. Para hacer uso de ellos simplemente teníamos que incluirlos con require:

```
var http = require("http");
```

Los paquetes o módulos que no están incluidos en forma nativa en NodeJS, los tenemos que incluir a través del gestor de paquetes npm.

El gestor de paquetes npm es un poquito distinto a otros gestores de paquetes que podemos conocer, como puede ser el de Linux, Fedora o Red Hat, por nombrar algunos, porque los instala localmente en los proyectos. Es decir, al descargarse un módulo, se agrega a un proyecto local, que es el que lo tendrá disponible para incluir. Aunque cabe decir que también existe la posibilidad de instalar los paquetes de manera global en nuestro sistema.

Para incluir un paquete tenemos que escribir en la línea de comandos posicionados en la ruta del proyecto a la cual queremos agregarle el paquete la instrucción npm install y el nombre del módulo que queremos instalar, por ejemplo:

```
npm install mime
```

Una vez que ejecutamos el programa npm indicando que se va a instalar el paquete 'mime' veremos que en nuestro directorio del proyecto se crea una carpeta llamada 'node_modules' (si instalamos un segundo paquete siempre se insertaran en dicha carpeta) y en dicha carpeta encontraremos el paquete 'mime' (con todo su código fuente).

En la carpeta ejercicio16 tenemos el mismo ejercicio10 reemplazando la variable que contenía los MIME types por el modulo mime.

Si se fijan en la carpeta verán que existe la carpeta node_modules con la carpeta mime dentro donde está el código del paquete.

Para usar un paquete es muy común que los desarrolladores incluyan una documentación para su uso.

Podemos entrar a la carpeta mime y abrir el archivo README.md donde explica cómo podemos utilizarlo.

Entonces el código modificado reemplaza el objeto literal que habíamos definido por la inclusión del módulo mime.

```
var mime=require('mime');
```

La instrucción require() recibe como parámetro el nombre del paquete que queremos incluir e inicia una búsqueda en el sistema de archivos, en la carpeta "node_modules" y sus hijos, que contienen todos los módulos que podrían ser requeridos.

Otra parte que cambia el código de nuestro programa es cuando tenemos que obtener el tipo mime del recurso a devolver al navegador:

```
var tipo=mime.lookup(camino);
console.log(tipo);
respuesta.writeHead(200, {'Content-Type': tipo});
```

Llamamos a la función lookup indicando el recurso y nos devuelve el tipo mime que le corresponde.

Podemos probar de tipear el parámetro npm ls en la línea de comandos en la carpeta de nuestro proyecto y veremos todos los paquetes que contiene nuestro proyecto (en nuestro ejemplo solo uno 'mime'), luego del @ aparece el número de versión:

```
C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 2\ejemplos\eje
rcicio16>npm ls
C:\Users\MariaVeronica\Documents\UTN-Desarrollo con NodeJS\Unidad 2\ejemplos\eje
rcicio16
└── mime@1.3.4
```

Otras instrucciones posibles de npm son la de publicar paquetes (con "publish"), instalar globalmente (poniendo el flag -g al hacer el "install"), desinstalar, incluso premiar (puntuar paquetes hechos por otras personas), etc.

Por último, sobre npm, cada paquete tiene entre su código un archivo package.json que contiene en notación JSON los datos del paquete en sí. Es como una tarjeta de identificación del paquete, que puede servir para informarte a ti mismo y a cualquier sistema informático de sus características. Si tú fueras el creador de un paquete, o crearas alguna aplicación con NodeJS, también deberías incluir "package.json" con los datos del módulo o aplicación que se está creando, como son el autor, versión, dependencias, etc.

Instalar paquetes de manera global con npm

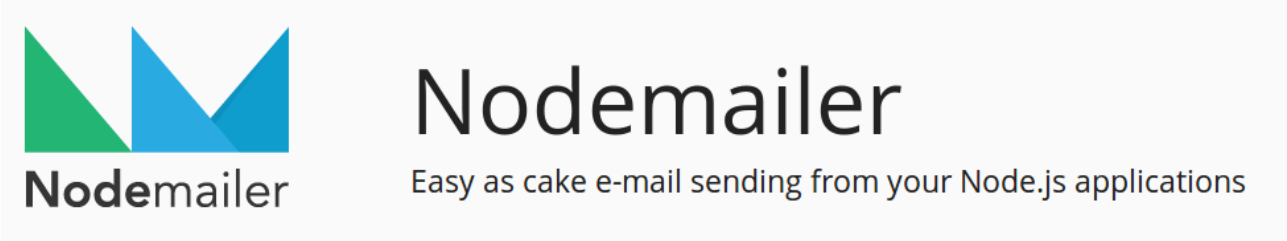
Como se ha dicho antes, npm instala los paquetes para un proyecto en concreto, sin embargo existen muchos paquetes de Node que te facilitan tareas relacionadas con el sistema operativo. Estos paquetes, una vez instalados, se convierten en comandos disponibles en terminal, con los que se pueden hacer multitud de cosas. Existen cada vez más módulos de Node que nos ofrecen muchas utilidades para los desarrolladores, accesibles por línea de comandos, como Bower, Grunt, etc.

Las instrucciones para la instalación de paquetes de manera global son prácticamente las mismas que ya hemos pasado para la instalación de paquetes en proyectos. En este caso simplemente colocamos la opción "-g" que permite que ese comando se instale de manera global en tu sistema operativo.

```
npm install -g grunt-cli
```

Ese comando instala el módulo grunt-cli de manera global en tu sistema.

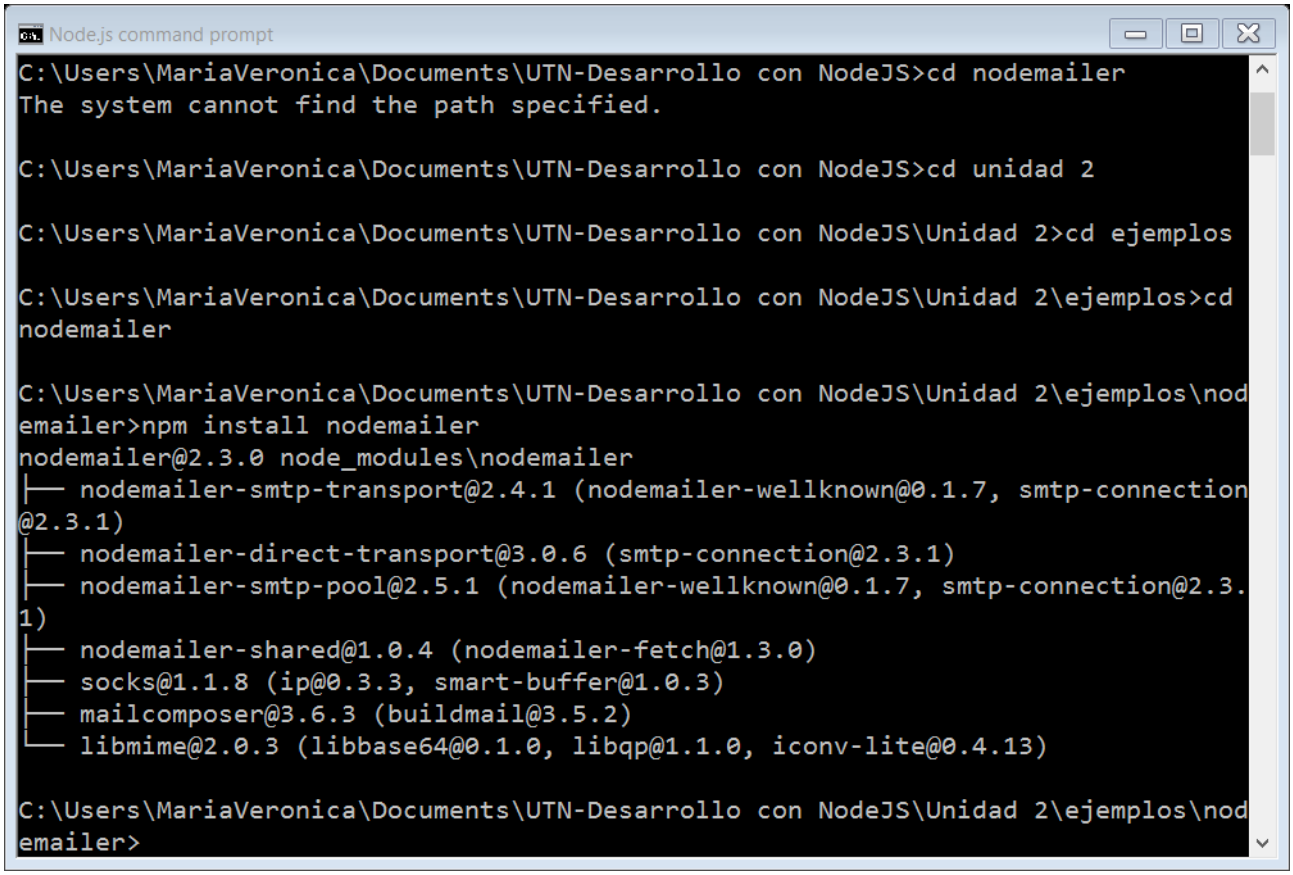
Ejemplo con Nodemailer



Vamos a hacer un ejemplo utilizando el módulo nodemailer.js que nos permite hacer el envío de mails de manera muy sencilla. Pueden encontrar información sobre este paquete en ***https://www.npmjs.com/package/nodemailer*** y en ***http://nodemailer.com/***

Primero que nada creamos una carpeta nodemailer en la cual instalamos el paquete a través de npm install nodemailer.

Una vez hecho esto en la consola veremos la información del paquete instalado y su versión:



Luego creamos nuestro archivo nodemailer.js en el que escribimos el código necesario para poder enviar mails.

Primero que nada necesitamos un objeto transporte:

var transporter = nodemailer.createTransport(transport[, defaults])

Donde:

- transporter es el objeto capaz de enviar el mail
- transport es la configuración del objeto de transporte, connection url or a transport plugin instance
- defaults valores por defecto para el envío

Una vez que tenemos el objeto transporte podemos hacer envío de mails a través de:

transporter.sendMail(data[, callback])

Donde:

- data es el contenido del mail a enviar
- callback es una función opcional a ejecutar una vez que el mensaje se entregó o falló

El código completo de nuestro ejercicio es el siguiente :

```
var nodemailer = require("nodemailer");

var smtpTransport = nodemailer.createTransport('smtps://user%40gmail.com:pass@smtp.gmail.com');

smtpTransport.sendMail({
   from: "My Name <me@example.com>", // sender address
   to: "Your Name <you@example.com>", // comma separated list of receivers
   subject: "Hello ✔", // Subject line
   text: "Hello world ✔" // plaintext body
}, function(error, response){
   if(error){
      console.log(error);
   }else{
      console.log("Message sent: " + response.message);
   }
});
```

Haciendo uso de Gmail. Para probar el ejemplo deben reemplazar user y pass por su usuario y password de Gmail y las direcciones del from y el to.

Una aclaración más, para poder usar Gmail de esta forma para probar el envío de mails. A pesar de que Gmail es la manera más rápida de comenzar con el envío de mensajes de correo electrónico, quizá no es la mejor por las trabas de seguridad. Gmail espera que el usuario sea un usuario real y no un robot, por lo que ejecuta una gran cantidad de heurísticas para cada intento de inicio de sesión y bloquea cualquier cosa que parezca sospechosa para defender al usuario de intentos de robo de cuentas.

Además Gmail creo el concepto de aplicaciones "Less Secure", que es, básicamente, cualquier persona o aplicación que utilice la contraseña normal para iniciar sesión en Gmail. Un nombre de usuario puede enviar correo (apoyo a las aplicaciones "less secure» habilitado), pero otro es bloqueado (apoyo a las aplicaciones "less secure" está desactivada). Puede configurar su cuenta de Gmail para permitir que las aplicaciones menos seguras aquí: ***https://www.google.com/settings/security/lesssecureapps***

Otra alternativa para el envío de mails en lugar de utilizar Gmail es XOAUTH2 para evitar problemas de acceso. Pueden consultar más información aquí: ***http://nodemailer.com/2-0-0-beta/using-oauth2/***

Ejemplo de upload de archivos con modulo formidable

Una actividad muy común en un sitio web es poder enviar desde un cliente (navegador) un archivo para que sea almacenado en el servidor.

Como Node.js es de bastante bajo nivel el implementar todo el código para procesar los datos que llegan desde un formulario html que adjunta uno o más archivos es bastante complejo, así que, de la misma forma que hemos utilizado el paquete nodemailer.js, vamos a utilizar un módulo desarrollado por la comunidad de Node.js que nos simplifica esta tarea llamado formidable.

Pueden consultar las características del módulo y consultar información aquí: ***https://github.com/felixge/node-formidable***.

Crearemos nuestro archivo js y las páginas correspondientes en la carpeta public. Por supuesto tendremos que instalar en la carpeta de nuestro proyecto el modulo formidable, a través de npm install formidable, lo que nos dejara la carpeta node_modules con el contenido del paquete dentro.

El código de este ejemplo lo encontraran en la carpeta formidable.

No hay mucho que explicar con respecto a las páginas index.html y formulario.html solo aclarar que la etiqueta del formulario tiene que contener el atributo enctype con el valor multipart/form-data para indicar que el formulario adjunta archivos.

Con respecto a nuestro archivo formidable.js en la función subir esta toda la lógica que tenemos que emplear para realizar el upload. Encontrarán el código del ejemplo comentado con su funcionamiento.

En la función listar tenemos la lógica correspondiente al listado de los archivo subidos. También encontraran comentado el código.

Comunicación con Mysql utilizando el módulo mysql

Vamos a ver cómo podemos desde nuestra aplicación Node.js comunicarnos con un servidor de base de datos Mysql.

Para ello tenemos que tener instalado el servidor de base de datos y ello lo conseguimos de forma sencilla instalando xampp o Wampserver.

Vamos a ir a phpmyadmin y creamos la base de datos que llamaremos basenode. Nuestra aplicación se encargara de crear dentro de la base una tabla de usuarios, de cargar registros y de listar los registros de la misma.

Node.js deja liberada a la comunidad para implementar módulos para comunicarse con otras aplicaciones. Para comunicarnos con MySQL existe un módulo llamado 'mysql'.

Creamos la carpeta mysql donde colocaremos los archivos correspondientes a nuestra aplicación y dentro de ella instalamos el modulo a través de npm install mysql.

Creamos la carpeta public con los html estáticos y el archivo mysql.js.

El código del archivo mysql.js esta comentado para que comprendan su funcionamiento.

A partir de este ejemplo podríamos pensar en crear una aplicación CRUD un poco más atractiva sobre todo visualmente, hemos visto como servir paginas html desde Node.js y en ese caso podemos incluir un diseño más interesante. Pero que sucede si dentro de la página queremos mostrar el listado de los productos por ejemplo. Como hacemos para recuperar los datos desde Node y enviárselos al html para que los muestre dentro de la misma.

En nuestro ejemplo la función listado arma el html e incluye la variable datos dentro del mismo. Como haríamos si por un lado tenemos un html con su css, JQuery, imágenes, etc. O con algún framework como Bootstrap y queremos enviar la variable datos para incluirla dentro de los datos a mostrar.

```
function listado(respuesta) {
    conexion.query('select codigo,descripcion,precio from articulos', function(error,filas){
        if (error) {
            console.log('error en el listado');
            return;
        }
        respuesta.writeHead(200, {'Content-Type': 'text/html'});
        var datos='';
        for(var f=0;f<filas.length;f++){
            datos+='Codigo:'+filas[f].codigo+'<br>';
            datos+='Descripcion:'+filas[f].descripcion+'<br>';
            datos+='Precio:'+filas[f].precio+'<hr>';
        }
        respuesta.write('<!doctype html><html><head></head><body>');
        respuesta.write(datos);
        respuesta.write('<a href="index.html">Retornar</a>');
        respuesta.write('</body></html>');
        respuesta.end();
    });
}
```

Para ello vamos a ayudarnos de Express y Jade como veremos en la siguiente Unidad. Express como ya dijimos es un framework de Node.js que lo que hace es crearte una estructura de trabajo para darte las funcionalidades más repetitivas ya servidas. Express incorpora en forma automática el sistema de plantillas Jade.

Nodemon

Un pequeño problema que tiene NodeJS es que cada vez que actualizamos algo en el código js debemos reiniciar el servidor, pero como no puede ser de otra forma, eso ya no es un problema, ya que tenemos ***nodemon***, otro paquete que hace el inicio y reinicio del server cada vez que hagamos algún cambio sin apenas enterarnos de forma automática. Si no tuviéramos instalado nodemon nuestra aplicación la correríamos tipeando en la pantalla:

```
node app.js
```

y cada vez que hiciéramos un cambio en nuestro código tendríamos que bajar y subir nuevamente el servidor.

Con nodemon nos olvidamos de hacer esto. Nodemon es una utilidad que hará un seguimiento de cualquier cambio en nuestro código y reiniciara automáticamente el servidor.

Su instalación es tan sencilla como dirigirse a la terminal y escribir el siguiente comando:

```
npm install -g nodemon
```

Hacemos la instalación a nivel global, para eso utilizamos el parámetro –g.

Una vez instalado correctamente nodemon, para utilizarlo en nuestro proyecto simplemente debemos escribir lo siguiente en la terminal situados en la raíz del proyecto:

```
Nodemon app.js
```

En lugar de node app.js.

Lo que vimos:

El módulo http, como interpreta los pedidos del navegador, como servir archivos estáticos html, archivos css, jpg, mp3, mp4, etc y como tomar datos de un formulario o de una url con GET.

El gestor de paquetes npm de NodeJS y el uso de algunos paquetes a través de ejemplos.

Lo que viene:

En la próxima unidad veremos cómo incorporar Express y Jade en nuestros proyectos.

Bibliografía

- Alex Young y Marc Harter (2014). Node.js in practice. Manning.
- Barsarat Ali Syed (2014). Beggining Node.js. Apress.
- Ethan Brown (2014). Web Development with Node & Express. O'Reilly.
- Gimson, Matthew. (2015). Practical Guide for Begginers. Kindle Edition.

- Sitios web de referencia y consulta:
- *https://nodejs.org/api/index.html*
- *https://www.npmjs.com/*
- *http://www.desarrolloweb.com/manuales/manual-nodejs.html*
- *http://www.javascriptya.com.ar/nodejsya/*