

16 - Framework Express - Instalación

Hemos visto hasta ahora que con Node.js podemos desarrollar un sitio web completo, servir páginas estáticas y dinámicas.

La comunidad de Node.js ha desarrollado un framework para implementar sitios web llamado Express. Este framework nos facilita y nos ordena el desarrollo de sitios web.

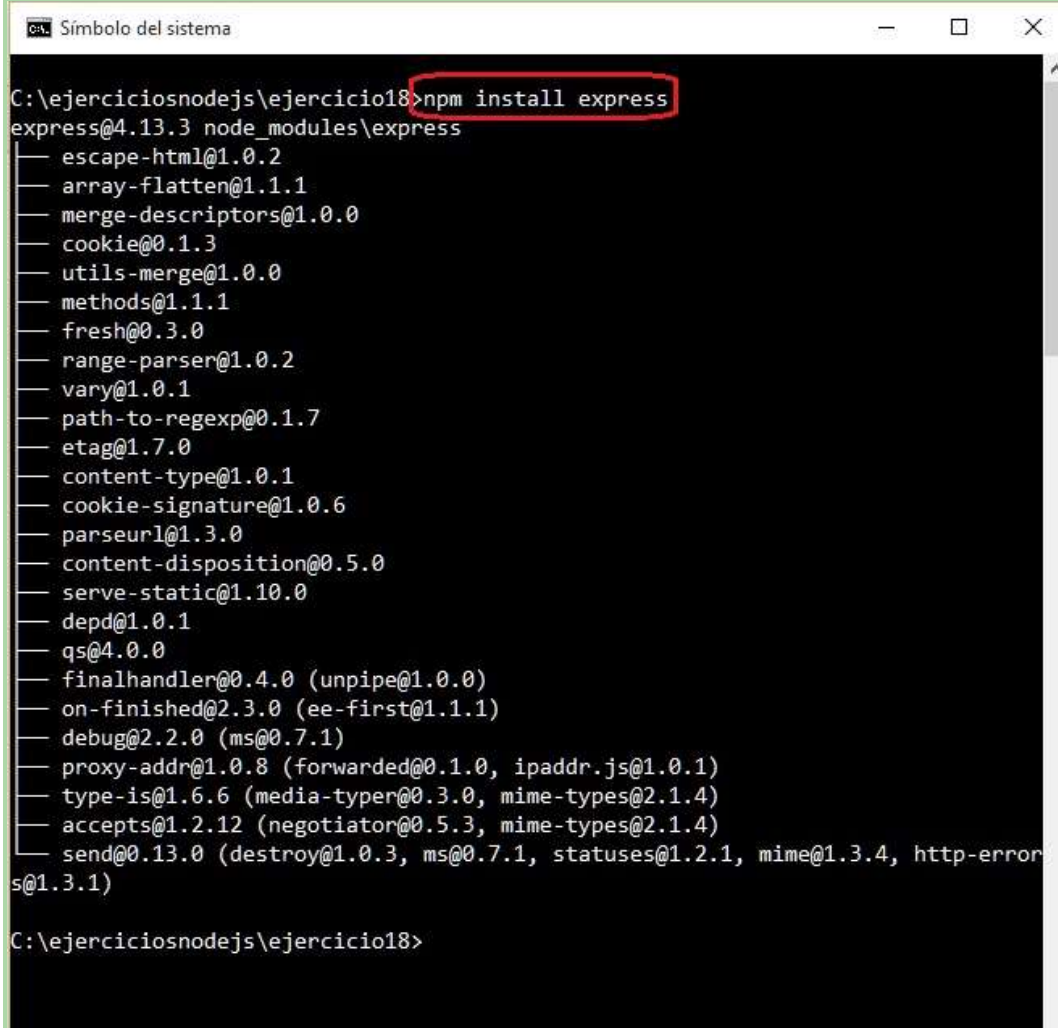
Para trabajar con Express lo primero que haremos es instalarlo en nuestro proyecto. El Framework Express está constituido por un conjunto de módulos.

Instalación

Lo primero que haremos es crear una carpeta : ejercicio18.

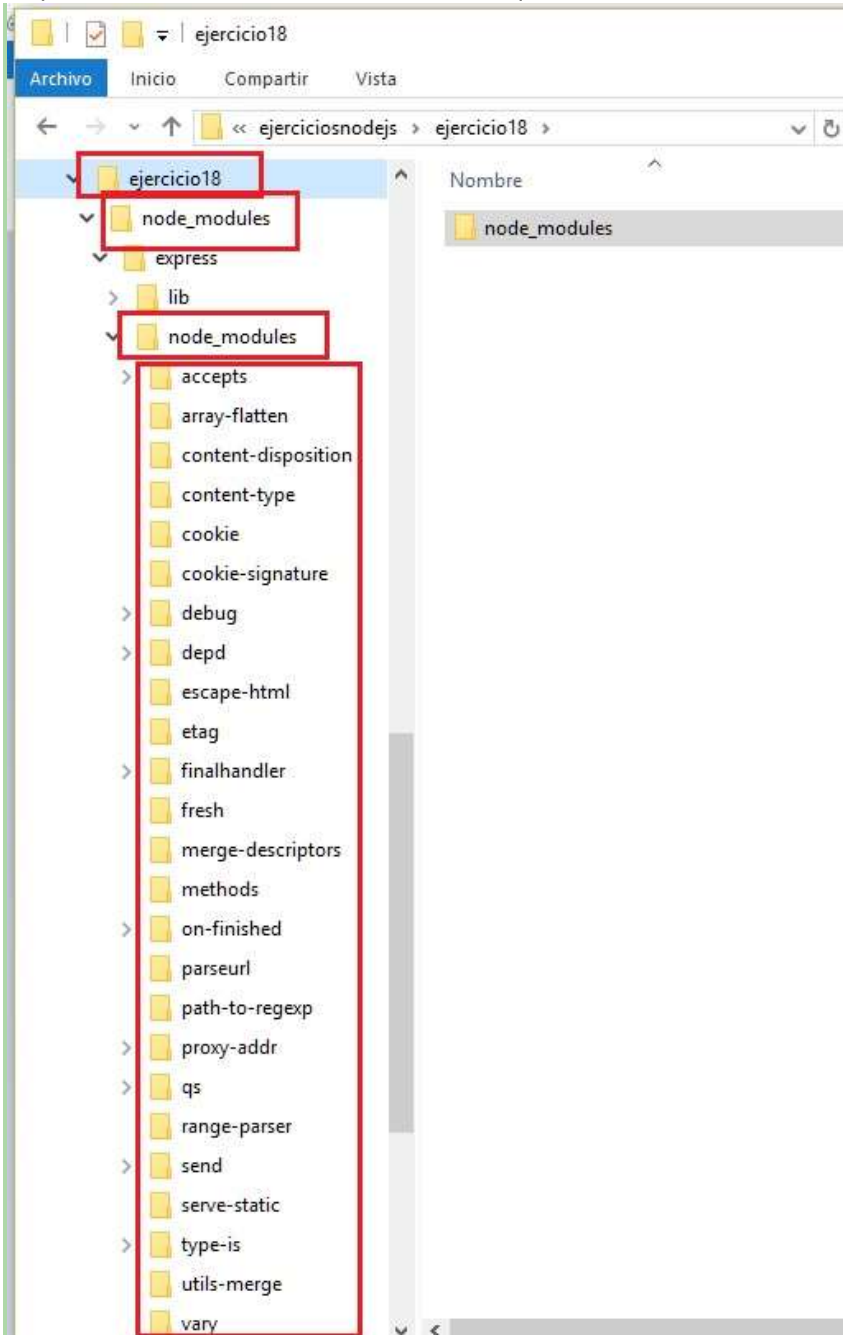
Una vez creada la carpeta desde la línea de comandos procedemos a instalar el framework mediante el npm:

```
c:\ejerciciosnodejs\ejercicio18>npm install express
```



```
C:\ejerciciosnodejs\ejercicio18>npm install express
express@4.13.3 node_modules\express
├── escape-html@1.0.2
├── array-flatten@1.1.1
├── merge-descriptors@1.0.0
├── cookie@0.1.3
├── utils-merge@1.0.0
├── methods@1.1.1
├── fresh@0.3.0
├── range-parser@1.0.2
├── vary@1.0.1
├── path-to-regexp@0.1.7
├── etag@1.7.0
├── content-type@1.0.1
├── cookie-signature@1.0.6
├── parseurl@1.3.0
├── content-disposition@0.5.0
├── serve-static@1.10.0
├── depd@1.0.1
├── qs@4.0.0
├── finalhandler@0.4.0 (unpipe@1.0.0)
├── on-finished@2.3.0 (ee-first@1.1.1)
├── debug@2.2.0 (ms@0.7.1)
├── proxy-addr@1.0.8 (forwarded@0.1.0, ipaddr.js@1.0.1)
├── type-is@1.6.6 (media-typer@0.3.0, mime-types@2.1.4)
├── accepts@1.2.12 (negotiator@0.5.3, mime-types@2.1.4)
├── send@0.13.0 (destroy@1.0.3, ms@0.7.1, statuses@1.2.1, mime@1.3.4, http-errors@1.3.1)
└──
```

Cuando se instala el framework como vemos se crea la carpeta 'node_modules' y en esta una subcarpeta 'express'. Si entramos en la carpeta 'express' veremos que también tiene su carpeta 'node_modules' con todos los módulos que depende el framework express:



Ya tenemos el framework de Express instalado, ahora procedamos a crear una aplicación mínima.

En la carpeta ejercicio18 procedamos a crear un archivo llamado 'app.js' y en su interior:

```
var express=require('express');
var app=express();

app.get('/',function (req,res){
    res.send('<!doctype html><html><head></head><body><h1>'+
        'Mi primer pagina</h1></body></html>');
});

var server=app.listen(8888,function(){
    console.log('Servidor web iniciado');
```

```
});
```

En principio como sabemos no podemos sacar una conclusión de las ventajas que propone un framework implementando un sitio que retorna una página (un framework tiene ventajas cuando implementamos programas de mediana o gran complejidad)

Veamos como funciona nuestro programa. Primero requerimos el módulo 'express' que retorna la referencia a una función.

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

Llamamos a la función que nos retorna un objeto que se almacena en la variable app:

```
var app=express();
```

Mediante el método get del objeto app procedemos a especificar que para la ruta '/' (raíz de nuestro sitio) ejecute la función anónima que le pasamos como segundo parámetro.

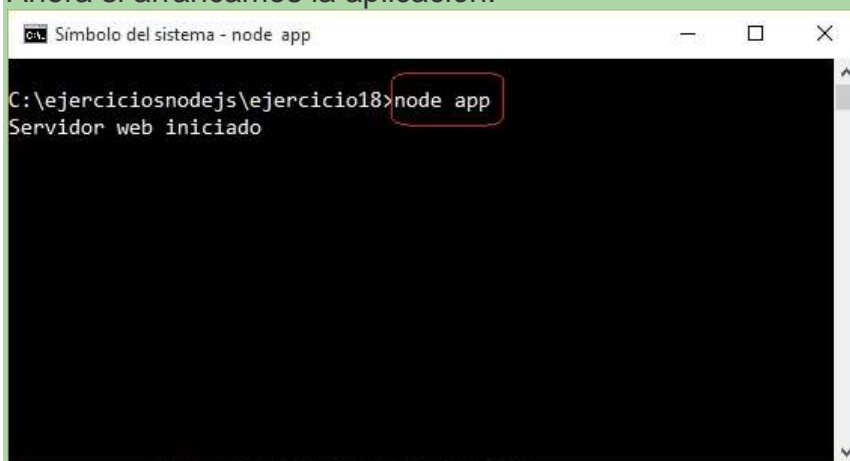
La función anónima recibe dos objetos 'req' y 'res'. Mediante el objeto 'res' respondemos al navegador que hizo la solicitud enviando código HTML:

```
app.get('/',function (req,res){
    res.send('<!doctype html><html><head></head><body><h1>'+
        'Mi primer pagina</h1></body></html>');
});
```

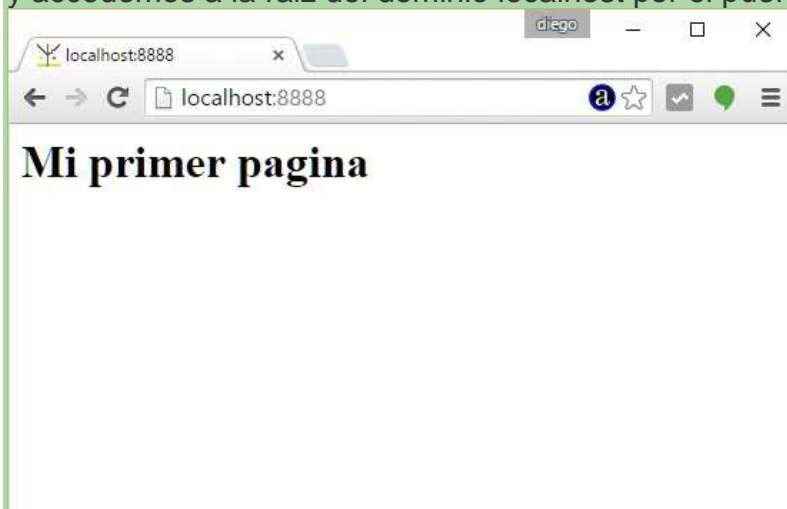
Por último para arrancar el servidor web debemos llamar al método listen:

```
var server=app.listen(8888,function(){
    console.log('Servidor web iniciado');
});
```

Ahora si arrancamos la aplicación:



y accedemos a la raíz del dominio localhost por el puerto 8888 tenemos como resultado:



Cada vez que implementemos un proyecto que requiera el framework Express lo primero que deberemos hacer es instalarlo para dicho proyecto.

17 - Framework Express - servir archivos estáticos html, css, jpg, mp3, mp4, ico, js etc

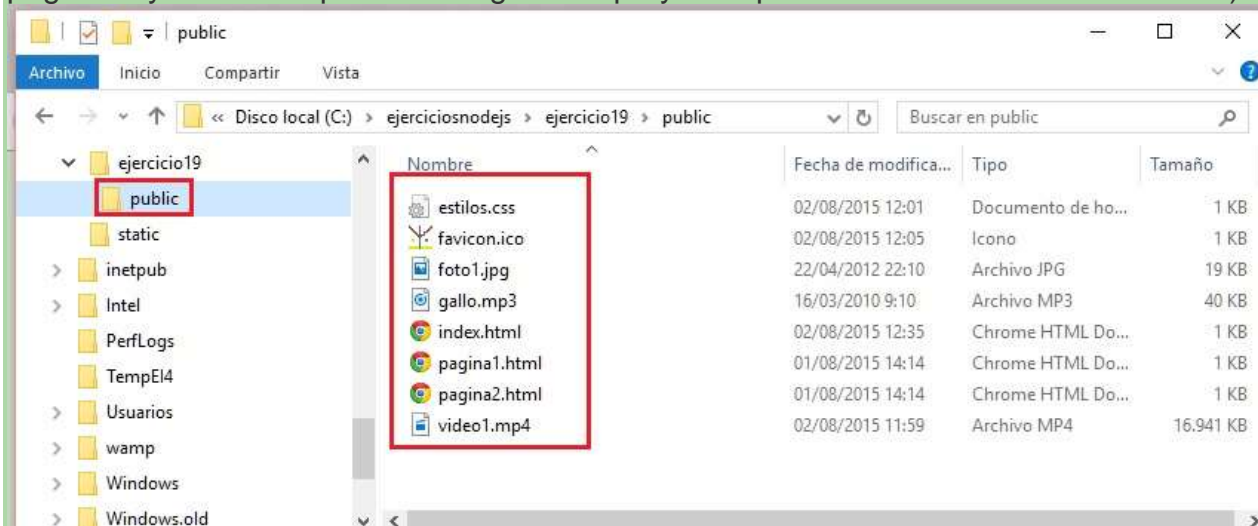
En el concepto anterior vimos los pasos para implementar una aplicación web mínima utilizando el Framework Express.

Ahora vamos a ver que debemos agregarle a dicha aplicación para que nuestro servidor retorne al navegador archivos estáticos con distintos formatos como podrían ser imágenes, archivos HTML, js, videos mp4 etc.

Vamos a implementar el mismo problema que resolvimos anteriormente con Node.js sin utilizar un framework.

Problema

Confeccionaremos un sitio que contenga una serie de archivos html, css, jpg, mp3, mp4 e ico. Crearemos un directorio llamado 'ejercicio19' y dentro de este otro directorio interno llamado 'public' donde se deben disponer todos los archivos html,css, jpg, mp3, mp4 e ico (al final de la página hay un enlace para descargar este proyecto que contiene todos estos archivos):



Ahora procedemos a instalar el framework Express para nuestro nuevo proyecto (tenemos en cuenta que en casa proyecto debemos instalarlo)

Para la instalación vimos que desde la línea de comando nos ubicamos en la carpeta de nuestro proyecto (ejercicio19) y utilizamos la herramienta npm:

```
c:\ejerciciosnodejs\ejercicio19>node install express
```

Ya tenemos instalado la última versión del framework Express y pasamos a crear el archivo app.js:

```
var express=require('express');
var app=express();

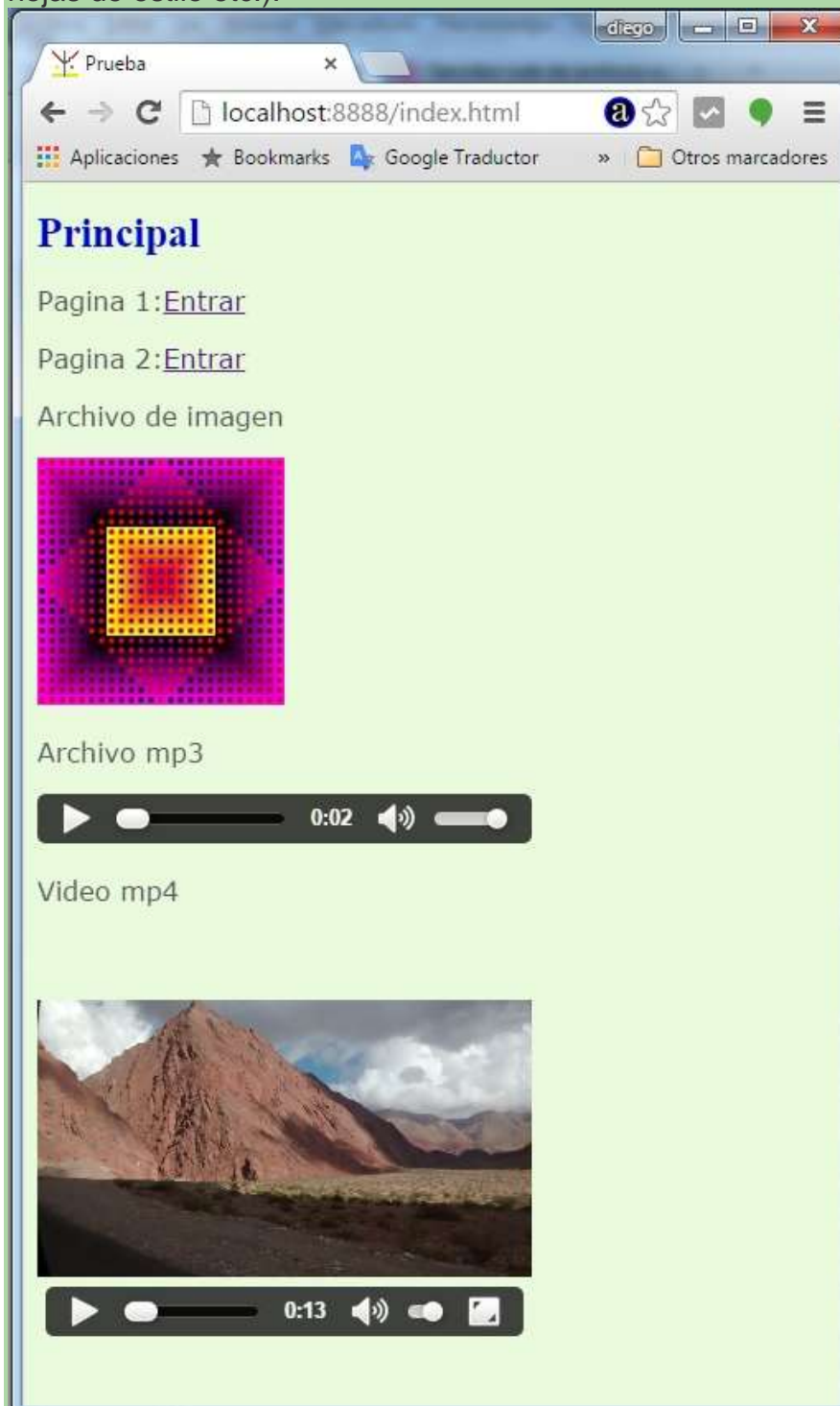
app.use(express.static(__dirname + '/public'));

var server=app.listen(8888,function(){
    console.log('Servidor web iniciado');
});
```

Ya podemos arrancar nuestro programa desde la línea de comandos estando en el directorio de nuestro proyecto:

```
node app
```

Desde el navegador hacemos las peticiones al servidor local y podemos comprobar que nos retorna los distintos recursos enumerados dentro de cada página (imágenes, audios, videos, hojas de estilo etc.):



Si vemos para poder servir archivos estáticos solo tenemos que llamar al método use y dentro llamar al método static de la variable express con un string que indique el path donde se encuentran los archivos estáticos.

La variable global '__dirname' almacena el path donde se encuentra la aplicación Node.js propiamente dicha, en nuestro ejemplo se encuentra en c:\ejerciciosnodejs\ejercicio19

Le concatenamos la subcarpeta donde se almacenan los archivos estáticos (recordemos que creamos una carpeta llamada public donde localizamos los archivos estáticos):

```
app.use(express.static(__dirname + '/public'));
```

18 - Framework Express - recuperar datos de formulario (POST) y parámetros url (GET)

Express lo definen sus autores como un framework minimalista, esto hace que muchas de las actividades de nuestra aplicación debemos codificarlas o utilizar otros módulos de terceros. Para recuperar los datos de un formulario HTML o los parámetros de una url debemos agregar además del framework Express un módulo para parsear los datos que llegan del navegador. Hay muchos módulos que hacen esta actividad el más común es el módulo 'body-parser'.

Problema

Confeccionar una aplicación que permita ingresar dos enteros por teclado. Luego al presionar el botón submit generar una página dinámica mostrando todos los números comprendidos entre el primer valor y el segundo de uno en uno. Los números mostrarlos como hipervínculos que al ser presionados generen una página dinámica con la tabla de multiplicar del valor seleccionado. Primero crearemos la carpeta donde alojaremos nuestro proyecto: ejercicio20.

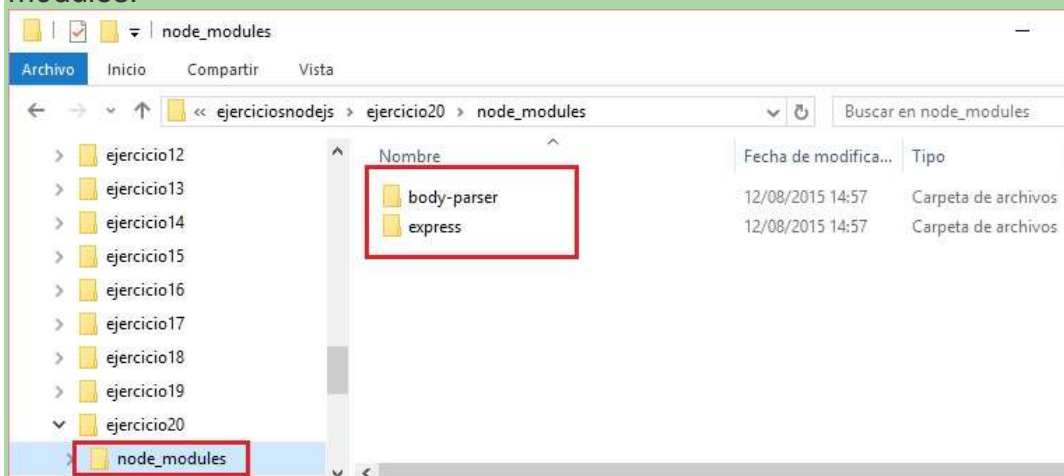
Ya tenemos la carpeta ejercicio20 creada ahora nos posicionamos en dicha carpeta y procedemos a instalar el framework 'express' y el módulo 'body-parser':

```
c:>ejerciciosnodejs\ejercicio20\npm install express
```

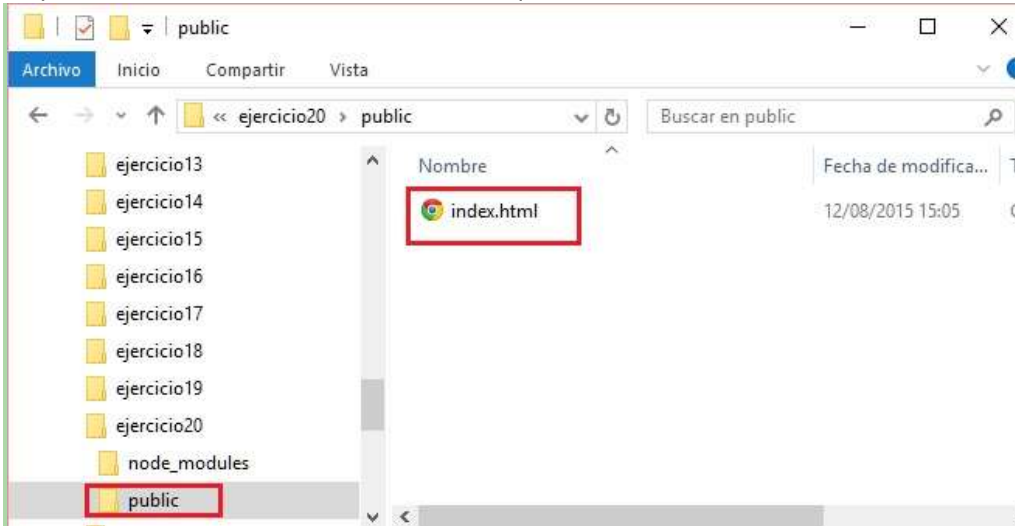
Y luego:

```
c:>ejerciciosnodejs\ejercicio20\npm install body-parser
```

En realidad el orden en que instalamos los módulos es indistinto pero luego de esto si vemos el contenido de la carpeta node_modules veremos que tiene las dos carpetas con los respectivos módulos:



Ahora procedemos a crear la carpeta public dentro de la carpeta ejercicio20 donde dispondremos la página estática HTML donde hay un formulario que pide el ingreso de dos números:



Su contenido es:

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Formulario</title>
</head>
<body>
  <p>Mostrar todos los numeros comprendidos entre los dos
valores ingresados de uno en uno.</p>
  <form action="mostrarnumeros" method="post">
    Ingrese numero inferior:
    <input type="text" name="numero1" size="10"><br>
    Ingrese numero superior:
    <input type="text" name="numero2" size="10"><br>
    <input type="submit" value="Mostrar">
  </form>
</body>
</html>
```

Creamos finalmente la aplicación de Node.js para servir páginas estáticas, recuperar datos del formulario HTML y finalmente recuperar datos de parámetros de hipervínculos.

La aplicación la creamos en la carpeta 'ejercicio12' y le damos como nombre app.js:

```
var express = require('express');
var app=express();
var bodyParser = require('body-parser');

//especificamos el subdirectorio donde se encuentran las
páginas estáticas
app.use(express.static(__dirname + '/public'));

//extended: false significa que parsea solo string (no
archivos de imagenes por ejemplo)
```

```
app.use(bodyParser.urlencoded({ extended: false }));

app.post('/mostrar numeros', function (req, res) {
  var num1=req.body.numero1;
  var num2=req.body.numero2;
  num1=parseInt(num1);
  num2=parseInt(num2);
  var pagina='<!doctype html><html><head></head><body>';
  for(var x=num1;x<=num2;x++)
    pagina += '<a
href="/mostrartabla?valor='+x+'">'+x+'</a>'+ ' - ' ;
  pagina += '</body></html>';
  res.send(pagina);
})

app.get('/mostrartabla', function (req, res) {
  var num=req.query.valor;
  num=parseInt(num);
  var pagina='<!doctype html><html><head></head><body>';
  for(var x=1;x<=10;x++) {
    var tabla=num * x;
    pagina += num + ' * ' + x + ' = ' + tabla + '<br>';
  }
  pagina += '<a href="index.html">Retornar</a>';
  pagina += '</body></html>';
  res.send(pagina);
})

var server=app.listen(8888,function(){
  console.log('Servidor web iniciado');
});
```

Lo nuevo que aparece es que requerimos el módulo 'body-parser':

```
var bodyParser = require('body-parser');
```

Lo enlazamos con Express llamando al método use y pasando lo que devuelve la función `urlencoded`:

```
app.use(bodyParser.urlencoded({ extended: false }));
```

La página estática la retorna Express ya que registramos el directorio que debe servir páginas estáticas:

```
app.use(express.static(__dirname + '/public'));
```

Cuando presionamos el botón 'submit' de nuestro formulario la propiedad `action` tiene el valor `action="mostrar numeros"`, esto hace que nuestro programa capture dicha ruta con el siguiente código:

```
app.post('/mostrar numeros', function (req, res) {
  var num1=req.body.numero1;
```



```
var num2=req.body.numero2;
num1=parseInt(num1);
num2=parseInt(num2);
var pagina='<!doctype html><html><head></head><body>';
for(var x=num1;x<=num2;x++)
    pagina += '<a href="/mostrartabla?valor='+x+'">'+x+'</a>'+
- ' ';
pagina += '</body></html>';
res.send(pagina);
})
```

Es aquí donde recuperamos los dos controles del formulario mediante el objeto 'req' que tiene una propiedad 'body' y dentro de esta podemos acceder a todos los 'name' que definimos en el formulario HTML (si no utilizamos el módulo 'body-parser' luego la propiedad req.body tiene el valor undefined):

```
var num1=req.body.numero1;
var num2=req.body.numero2;
```

Los convertimos a entero los valores recuperados:

```
num1=parseInt(num1);
num2=parseInt(num2);
```

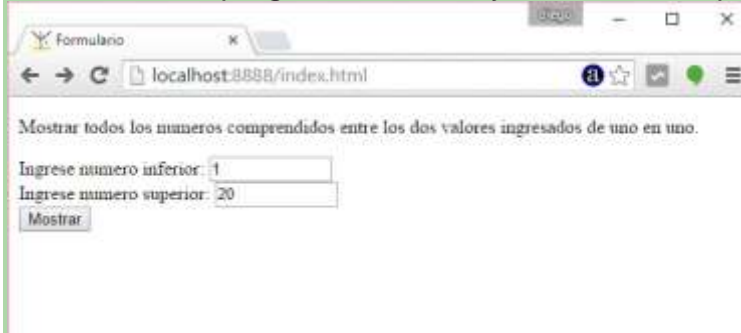
Y finalmente generamos la página en forma dinámica con todos los enlaces con cada número comprendido entre los valores ingresados de uno en uno:

```
var pagina='<!doctype html><html><head></head><body>';
for(var x=num1;x<=num2;x++)
    pagina += '<a href="/mostrartabla?valor='+x+'">'+x+'</a>'+
- ' ';
pagina += '</body></html>';
```

Enviamos al navegador la página generada dinamicamente:

```
res.send(pagina);
```

Si iniciamos el programa de node y solicitamos la página estática al navegador tenemos:



Y cuando presionamos el botón 'submit' podemos observar todos los valores comprendidos entre el primer y segundo valor ingresado:



Cada número que se muestra es un hipervínculo que tiene un parámetro llamado valor junto con el número propiamente dicho. Cuando el operador presiona el hipervínculo se captura el path del mismo con el siguiente código Node.js:

```
app.get('/mostrartabla', function (req, res) {  
    var num=req.query.valor;  
    num=parseInt(num);  
    var pagina='<!doctype html><html><head></head><body>';  
    for(var x=1;x<=10;x++) {  
        var tabla=num * x;  
        pagina += num + ' * ' + x + ' = ' + tabla + '<br>';  
    }  
    pagina += '<a href="index.html">Retornar</a>';  
    pagina += '</body></html>';  
    res.send(pagina);  
})
```

Acá de forma similar recuperamos el dato que viene en el hipervínculo mediante la propiedad 'query' del objeto 'req':

```
var num=req.query.valor;
```

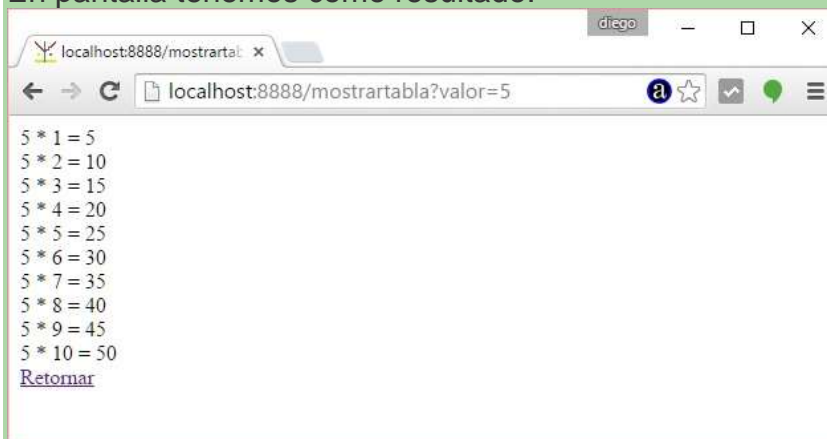
Luego de convertirlo a entero pasamos a generar la página HTML dinámica con la tabla de multiplicar de dicho valor:

```
num=parseInt(num);  
var pagina='<!doctype html><html><head></head><body>';  
for(var x=1;x<=10;x++) {  
    var tabla=num * x;  
    pagina += num + ' * ' + x + ' = ' + tabla + '<br>';  
}  
pagina += '<a href="index.html">Retornar</a>';  
pagina += '</body></html>';
```

Enviamos al navegador la página:

```
res.send(pagina);
```

En pantalla tenemos como resultado:



19 - Framework Express - Generador automático

El creador del framework Express ha implementado una aplicación (también esta aplicación está escrita en Node.js) que tiene por objetivo crear una estructura mínima de un proyecto que utilice Express (un conjunto de carpetas y archivos)

Si bien esto no es obligatorio como vimos en conceptos anteriores es muy beneficioso conocerla para que nosotros podamos definir como queremos organizar los archivos de un proyecto grande. El primer paso será instalar la aplicación 'express-generator'.

La aplicación 'express-generator' se ejecuta desde la consola de nuestro sistema operativo y no es un módulo que se instala en una aplicación particular. Para instalar una aplicación en Node.js que se la pueda ejecutar desde la línea de comandos también utilizamos el 'npm' pero le pasamos un parámetro extra '-g' (global), instalemos el programa express-generator:

```
npm install express-generator -g
```

Es muy importante no olvidarse del parámetro -g

Ahora que tenemos la aplicación 'express-generator' nos posicionamos en el directorio 'c:\ejerciciosnodejs' (o el directorio donde esta almacenando todos sus proyectos) y procederemos a crear nuestra primer aplicación Node.js utilizando Express y su generador de código:

```
c:\ejerciciosnodejs> express ejercicio21 --hbs
```

Estamos llamando al programa que acabamos de instalar que se llama 'express' y le pasamos dos parámetros, el primero indica el nombre de nuestro proyecto y el segundo el sistema de plantillas que utilizaremos para generar nuestras páginas dinámicas.

Ya tenemos creado la carpeta ejercicio21 y dentro de esta una serie de archivos y subcarpetas:

```
ejercicio21
  app.js
  package.json
  bin
    www
  public
    images
    javascripts
    stylesheets
  router
    index.js
    users.js
  views
    error.hbs
    index.hbs
    layout.hbs
```

Si observamos dentro de todas las carpetas que ha creado el generador de código 'express' no aparece la carpeta 'node_modules'. Es decir que todavía nos hace falta instalar el framework Express. Aquí es donde entra a jugar este nuevo archivo llamado package.json.

Si abrimos este archivo package.json:

```
{
  "name": "ejercicio21",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
}
```

```
"dependencies": {  
  "body-parser": "~1.13.2",  
  "cookie-parser": "~1.3.5",  
  "debug": "~2.2.0",  
  "express": "~4.13.1",  
  "hbs": "~3.1.0",  
  "morgan": "~1.6.1",  
  "serve-favicon": "~2.3.0"  
}
```

"name": Indicamos el nombre de proyecto

"version": Indicamos la versión del proyecto.

"start": Indicamos el archivo que se debe ejecutar para arrancar el proyecto

"dependencies": Indicamos todos los otros módulos que se requieren en nuestro proyecto.

Como podemos observar en "dependencies" aparece el framework 'express' y el módulo 'body-parser' que hemos utilizado en conceptos anteriores.

Ahora veremos que podemos instalar todos estos módulos con un único comando, nos posicionamos en el directorio ejercicio21 y ejecutamos:

```
c:\ejerciciosnodejs\ejercicio21>npm install
```

Cuando llamamos a 'npm install' sin ningún otro parámetro lo que hace es buscar el archivo 'package.json' y proceder a instalar todos los módulos especificados en la propiedad 'dependencies'.

Ahora ya tenemos creado la carpeta 'node_modules' con las 7 carpetas que coinciden con las dependencias especificadas en el archivo json:

```
body-parser  
cookie-parser  
debug  
express  
hbs  
morgan  
serve-favicon
```

Podemos ejecutar nuestra aplicación mínima creada con el 'express-generator':

```
c:\ejerciciosnodejs\ejercicio21>node ./bin/www
```

Tener en cuenta que el generador define como puerto el valor 3000 y no el 8888 que utilizamos en ejemplos anteriores.

Y ya podemos solicitar al servidor la página raíz del sitio (por el puerto 3000):

Ahora podemos ver otra forma de iniciar a nuestro proyecto en Node.js cuando definimos el archivo package.json:

En lugar de escribir:

```
c:\ejerciciosnodejs\ejercicio21>node ./bin/www
```

Escribimos:

```
c:\ejerciciosnodejs\ejercicio21>npm start
```

Recordemos que en el archivo json hay una propiedad start donde definimos el archivo que inicia nuestra aplicación:

```
"scripts": {  
  "start": "node ./bin/www"  
},
```

Hasta aquí hemos visto todo lo que podemos hacer en forma automática, en conceptos sucesivos veremos cada uno de los archivos y carpetas fundamentales para organizar un proyecto Node.js con 'Express'.