

**Real Time Web with Node.Js**

**Codeschool.com**

**1. Introducción a Node.Js**

# **What could you build?**

* Websocket Server (Like a chat server).
* Fast File Upload Client.
* Ad Server
* Any Real-Time Data Apps.

# **¿Qué es Node.Js?**

Node.Js es framework de Javascript non-blocking code que permite ejecutar varias tareas a la vez aunque alguna de ellas no haya terminado.

# **Blocking Code.**

Por ejemplo otros lenguajes que son blocking code, no pueden realizar una tarea hasta que la anterior haya acabado, por ejemplo:

1. Read file from Filesystem.
2. Print contents.
3. Do something else.

**var contents = fs.readFileSync(‘index.html’);**

**console.log(contents);**

**console.log(‘Doing something else’);**

# **Non-blocking Code.**

En cambio con un non-blocking code como node:

1. Read file from Filesystem.

* Cuando se haya completado, imprime su contenido. (esto es un callback).

1. Do something else. Continua a pesar de que el callback no haya finalizado.

**Fs.readFile(‘index.html’, function(err, contents){**

**console.log(contents);**

**}**

**console.log(‘Doing something else’);**

# Estructura

**var http = require(‘http’); // How we require modules**

**http.createServer(function(request, response){**

**response.writeHead(200); // Status code in header**

**response.write(“Hello, this is dog.”); // Response body**

**response.end(); // Close the connection**

**}).listen(8080); // Listen for connections on this port**

**console.log(‘Listening on port 8080…’);**

$ node hello.js // 🡪 Listening on port 8080

$ curl <http://localhost:8080> -> Hello, this is dog.

# Two callbacks

**var http = require(‘http’); // How we require modules**

**http.createServer(function(request, response){**

**response.writeHead(200); // Status code in header**

**response.write(“Do gis running.”);**

**setTimeout(function(){**

**response.write(“Dog is done.”);**

**response.end();**

**}, 5000);**

**}).listen(8080); // Listen for connections on this port**

# Typical blocking things

* Call out to web services.
* Read/Writes on the Database.
* Call to extensions.

**2. Events**

Muchos objetos en Node emiten eventos, los cuales son heredados del constructor EventEmitter.

Para personalizar dichos eventos en necesario requerir de la siguiente clase:

**var EventEmitter = require(‘events’).EventEmitter**

Ejemplo:

**var logger = new EventEmitter // error warn info (events)**

**logger.on(‘error’, function(message){**

**console.log(‘ERR: ‘ + message);**

**});**

**logger.emit(‘error’, ‘Spilled Milk’);**

# Alternate Syntax

// Option 1

**http.createServer(function(request, response){ … });**

// Option 2

**var server = http.createServer();**

**server.on(‘request’, function(request, response){… });**

**server.on(‘close’, function(){ .. });**

# 3.1 Streams

Los stream son como canales por donde es tranferida la información. Puedes ser de diferentes tipos: lectura, escritura o ambos.

**http.createServer(function(request, response)** // request es el canal de lectura

// response es el canal de escritura

**http.createServer(function(request, response){**

**response.writeHead(200);**

**request.on(‘readable’, function(){**

**var chunk = null;**

**while(null ¡== (chunk = request.read())){**

**console.log(chunk.toString());**

response.write(chunk);

**}**

**});**

**resquest.on(‘close’, function(){**

**response.end();**

**});**

**}).listen(8080)**

Node.Js tiene un método propio que permite realizar ambas operaciones juntas por lo que quedaría asi:

**http.createServer(function(request, response){**

**response.writeHead(200);**

**request.pipe(response);**

**}).listen(8080)**

# Reading and Writing a file

**var fs = require(‘fs’); // require filesystem module**

**var file = fs.createReadStream(“readme.md”);**

**var newFile = fs.createWriteStream(“readme\_copy.md”);**

**file.pipe(newFile);**

# Upload a File

Var fs = require(‘fs’);

Var http = require(‘http’);

**http.createServer(function(request, response){**

**var newFile = fs.createWriteStream(”readme\_copy.md”);**

**request.pipe(newFile);**

**request.on(‘end’, function(){**

**response.end(‘uploaded!’);**

**});**

**}).listen(8080);**

$ curl –upload-file readme.md <http://localhost:8080>

# File Uploading Progress

**http.createServer(function(request, response){**

**var newFile = fs.createWriteStream(”readme\_copy.md”);**

**var fileBytes = request.headers[‘content-length’];**

**var uploadedBytes = 0;**

**request.on(‘readable’, function(){**

**var chunk = null;**

**while(null ¡== (chunk = request.read())){**

**uploadedBytes += chunk.length;**

**var progress = (uploadedBytes / fileBytes) \* 100;**

**response.write(“progress: “ + parseInt(progress, 10) + “%\n”);**

**}**

**});**

**request.pipe(newFile);**

**…**

**}).listen(8080);**