

#### Sobre los Entrenadores

Luis Cairampoma

Luis es Desarrollador Fullstack y es parte del primer grupo con el que empezamos actividades en HackSpace.

Cristhian Huertos

Cristhian es Desarrollador Fullstack, apasionado por el monitoreo de condiciones en Sistemas Mecánicos & Hidráulicos

#### Programas Necesarios

[Node JS](https://nodejs.org/en/download/), [Git](https://git-scm.com/downloads" \t "_blank)

#### Semana 1 : Introduccion a Node.Js

**Ejemplos**Introduccion al backend y Node.js

Nociones sobre backend

El backend es la parte web del lado del servidor y base de datos, oculta al usuario. Este no lo puede observar pero lo puede usar indirectamente, ya que el backend trabaja con los datos que se envían desde el frontend como contraseñas al momento de autentificarte o como datos cuando registras tu tarjeta de crédito. Estos datos pueden ser registradas en una base de datos o pueden ser utilizados para hacer alguna consulta al servidor.

Los lenguajes o tecnologías más comunes a utilizar son Python con su framework Django, Rails de Ruby y Express de Node.js. Este último permite utilizar javascript como backend y es el que aprenderemos en este entrenamiento.

Introduccion a Node.js

NodeJS es un entorno de desarrollo para ejecutar JavaScript en el servidor, empresas como Uber, Netflix y Wikipedia usan Node. Es excelente para aplicaciones en tiempo real y hoy puedes aprender gratis.

Revisar este [link](http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js) para conocer mas sobre cuando usar Node.js

Instalacion de Node.js y su manejador de paquetes npm

Antes de instalar cualquier instalador verificar si existe alguna version de node instalada en nuestro entorno de desarrollo, eso se puede verificar corriendo el siguiente comando desde un terminal:

$ node --version

En caso de no tener Node instalado descargar el [instalador de Node.js para su sistema operativo](https://nodejs.org/en/download/) y seguir las instrucciones, otra alternativa para instalar Node.js si usa Linux o Mac es usar un manejador de versiones ([nvm](https://github.com/creationix/nvm)) el cual nos da una ventaja de poder usar diferentes versiones de node y ademas no necesitamos permisos de superusario para instalar librerias de node. 

# Instalar nvm

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.30.1/install.sh | bash

# Actualizar el bash\_profile

source ~/.bash\_profile

# Instalar la version estable de node y npm

nvm install stable

# Verificar que node y npm se instalaron correctamente

node --version

npm --version

Ejemplo 1

Una vez instalado podemos crear nuestro primer programa en node para eso en un editor de texto crear un nuevo archivo y guardarlo con el nombre holanode.js y escribir el siguiente codigo:

// holanode.js

function hola(){

console.log("Hola node");

}

hola();

Y para ejecutar el programa en el terminal ejecutamos el comando node holanode.js desde la carpeta donde guardamos el archivo

Ejemplo 2

Ejecutar codigo Javascript plano en el lado del servidor como el primer ejemplo no es de gran utilidad, es por eso que Node.js nos permite usar una serie de librerias (tambien conocidos como modulos) para hacer aplicaciones mas interesantes

En este ejemplo vamos a usar el modulo fs que viene incluido al instalar node el cual nos sirver para leer y escribir archivos externos

Vamos a crear una aplicacion que lee un archivo que tiene la siguiente estructura para contar el numero de votos

Angular 2

React 1

Jquery 1

Ember 1

React 5

Angular 3

Vanilla 2

Guardar el archivo anterior con el nombre votos.txt y crear guardar el siguiente archivo en el mismo directorio del archivo txt.

// Importar el modulo 'fs' de node

var fs = require('fs');

// Leer el archivo votos.txt y ejecutar una funcion cuando ah leido el archivo

fs.readFile('votos.txt', function (err, dataBuffer) {

// Convertir el archivo leido en una cadena de texto:

var dataTexto = dataBuffer.toString();

// Dividir el archivo de texto en un array que contiene cada linea del archivo

var lineas = dataTexto.split('\n');

// Inicializar una variable que contiene los resultados

var resultado = {};

// Iterar por cada linea del archivo

lineas.forEach(function(linea) {

// Dividir cada linea en palabras por separado

var palabras = linea.split(' ');

// Si la primera palabra tiene un valor

if (palabras[0]){

// Almacenar el nombre y el numero de votos en dos variables

var nombre = palabras[0];

var numero = parseInt(palabras[1]);

// Si es la primera vez que lee este nombre , agregar el numero de votos

if (!resultado[nombre]) {

resultado[nombre] = numero;

} else {

// Sino agregar al resultado que tiene el numero de votos de esta linea

resultado[nombre] += numero;

}

}

});

// Imprimir el resultado en la consola

console.log(resultado);

});

Y si ejecutamos el comando node contador.js deberiamos obtener el siguiente resultado en el terminal: { Angular: 5, React: 6, Jquery: 1, Ember: 1, Vanilla: 2 }

**Ejemplos**Instalacion de express y nuestro primer servidor web

El manejador de paquetes NPM

Como explicamos anteriormente node nos permite usar una serie de modulos en nuestra aplicacion, ademas de los modulos nativos que vienen incluidos (como el modulo 'fs' que usamos en el ejemplo anterior) podemos importar otros modulos para realizar distintas tareas, por ejemplo 'underscore', 'request', 'bluebird' entre otros. Pueden encontrar una lista de modulos en la web de [npm](https://www.npmjs.com/" \t "_blank)

Comenzando un proyecto

Para generar nuestro primer proyecto en node.js creamos una carpeta llamada 'miservidor' y desde el terminal ejecutamos el comando npm init y de manera opcional ingresamos los detalles de nuestro proyecto, de lo contrario presionar enter para rellenar estos detalles con los valores por defecto, al completar esta tarea podemos notar que se ha creado un archivo package.json en el directorio el cual contiene los detalles de nuestro proyecto y sus dependencias

A continuacion instalamos el modulo Express que sirve para crear un servidor web, desde el terminal ejecutamos el siguiente comando

npm install --save express

Nuestro primer servidor web

Y creamos un archivo con el nombre miserver.js

//miserver.js

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Mi primer servidor!');

})

app.listen(3000);

Al ejecutar el script anterior con el comando node miserver.js este inicia un servidor que esta escuchando los pedidos en el puerto 3000, el cual podemos verificar ingresando desde un navegador web a la ruta [http://localhost:3000](http://localhost:3000/) , para detener el servidor presionamos dos veces el comando Ctr+C en el terminal. 

Manejando las rutas

Express nos permite responder a cada consulta de manera especifica basada en la ruta de la cual se hace la llamada al servidor, de manera que si modificamos nuestro archivo miserver.js con el codigo siguiente y lo volvemos a ejecutar

//miserver.js

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Mi primer servidor!');

})

app.get('/perfil', function (req, res) {

res.send('Mi perfil!');

})

app.get('/amigos', function (req, res) {

res.send('Mis amigos');

})

app.listen(3000);

Ahora podemos ver que obtenemos diferentes respuestas al ingresar a rutas como<http://localhost:3000/perfil>

**Ejemplos**Creando un API en Node.js

Que es un API?

Un API es una forma de compartir los datos de una aplicación en internet. Por ejemplo, Twitter tiene un API que permite solicitar los tweets en un formato que facilita la utilizacion de estos en otra aplicación. Este es un gran ventajas de usar APIs , ser capaz de mezclar datos de distintas aplicaciones en una propia aplicación, o crear una aplicación que mejora la experiencia de uso de la aplicación de otra persona.

Nuestro primer api en node

En este ejemplo vamos a crear un 'endpoint' que nos va a retornar una lista de usuarios con algunos datos basicos que luego pueden ser usados por el frontend mediante una peticion ajax para mostrarlos en el sitio web con su respectivo formato. Para esto necesitamos instalar el modulo 'faker' de node que nos va ayudar a generar data aleatoria de todos los campos que necesitamos, desde el terminal en la carpeta que generamos nuestro proyecto anterior ejecutamos el comando npm install --save faker

var express = require('express');

var app = express();

var faker = require('faker');

app.get('/', function (req, res) {

res.send('Mi primer servidor!');

})

app.get('/friends', function (req, res) {

var randomName = faker.name.findName();

var randomEmail = faker.internet.email();

var randomImage = faker.image.avatar();

var usuario = {

nombre: randomName,

email: randomEmail,

imagen: randomImage

}

res.json(usuario);

})

app.get('/amigos', function (req, res) {

res.send('Mis amigos');

})

app.listen(3000);

La funcion implementada en el codigo anterior retorna los datos aleatorios para un usuario pero necesitamos regresar una lista de varios usuarios, para eso creamos una funcion que retorne los datos de un usuario y usando el modulo lodash de node, creamos un array donde almacena el resultado de ejecutar esa funcion una cierta cantidad de veces. Como los demas modulos instalamos 'lodash' npm install --save lodash

var express = require('express');

var app = express();

var faker = require('faker');

var \_ = require('lodash');

var generarUsuario = function(){

var randomName = faker.name.findName();

var randomEmail = faker.internet.email();

var randomImage = faker.image.avatar();

return {

nombre: randomName,

email: randomEmail,

imagen: randomImage

}

}

app.get('/', function (req, res) {

res.send('Mi primer servidor!');

})

app.get('/friends', function (req, res) {

var cantidad = \_.random(5,10)

var usuarios = \_.times(cantidad, generarUsuario)

res.json(usuarios);

})

app.get('/amigos', function (req, res) {

res.send('Mis amigos');

})

app.listen(3000);

Si accedemos a <http://localhost:3000/friends>

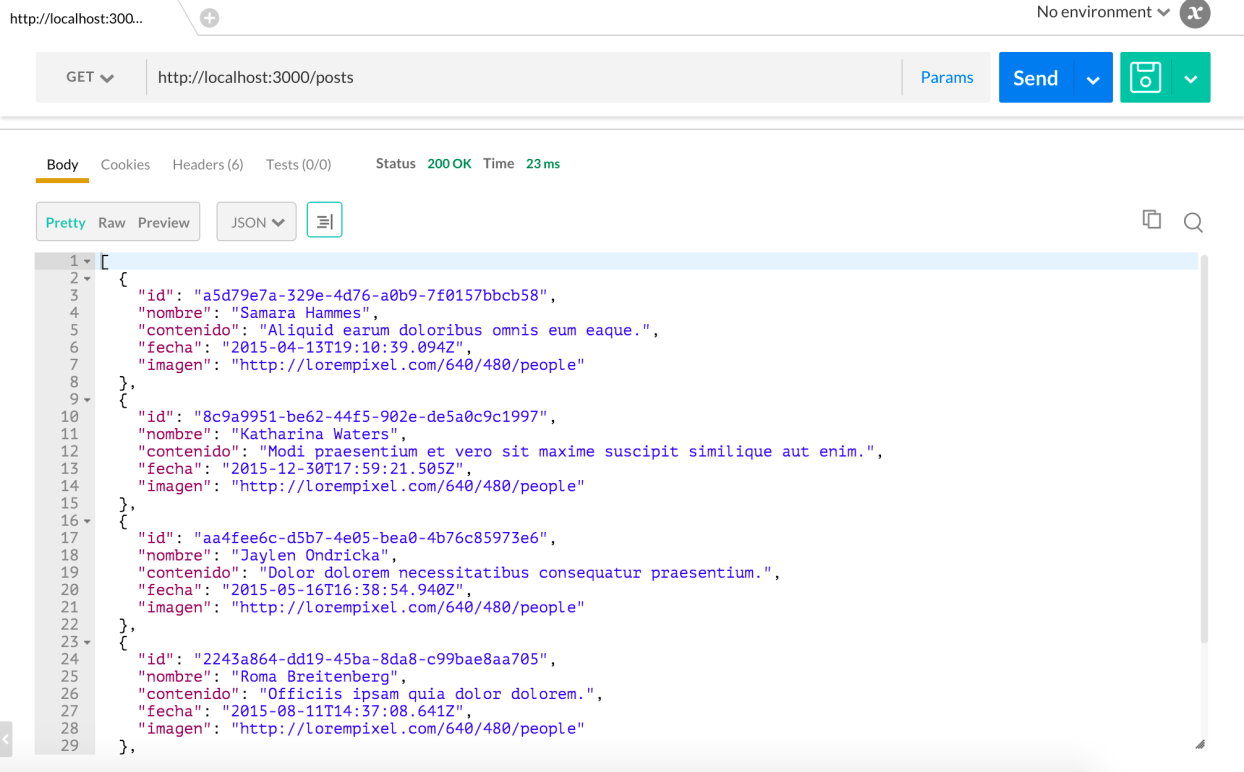


[S](http://hackspace.la/semana1/backend/reto-1.5.html)

**Reto 1**Primer reto del entrenamiento de Backend.

Reto 1 de Backend

El primer reto es crear un api en express con dos endpoints como se detalla en la imagenes



Para poder ver el resultado del api en un formato como se muestra en la imagen pueden descargar la extension de chrome [Postman](https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop) que sirve para hacer consultas http

Pueden ver la descripcion de todos los tipos de datos que genera la libreria Faker.js en su [repositorio](https://github.com/marak/Faker.js/)(el tipo de dato que utilizen no tiene que ser exacto, tambien pueden agregarle otros campos que crean convenientes).

#### Semana 2 : Introduccion a Hapi.Js

### ****Ejemplos****Introduccion al backend y Hapi.js

### Nociones sobre HapiJs

"Why Use HapiJs", “A rich framework for building applications and services hapi enables developers to focus on writing reusable application logic instead of spending time building infrastructure.”

HapiJs ha sido diseñado y desarrollado para crear de forma rápida servicios API REST a través de Javascript. Es muy simple y fácil de usar ya que tiene una sintaxis clara y una curva de aprendizaje rápida.

HapiJs ha demostrado ser sólido y estable a la hora de montarse en producción. Sitios Web como Walmart, han optimizado HapiJs en etapas de alto requerimiento de solicitudes por segundo (como el Black Friday). Este framework ha sido creado para funcionar en entornos empresariales donde varios equipos de trabajo utilizan piezas de código que se conectan entre diversos proyectos. Esta característica hace que utilizar HapiJs sea fácil de aplicar para la escritura y reutilización de código modular.

### Iniciando con Hapi.js

Para iniciar con Hapi.Js debemos tener instalado Node y NPM.

Si aún no tienes instalado, revisa este [Link](http://hackspace.la/semana1/backend/ejemplo-1.2.html)

Revisar este [link](http://hapijs.com/) para conocer mas sobre el desarrollo de HapiJs

### Instalacion de Hapi.js y creando nuestro primer servidor

Si Ud. cuenta con NodeJs y NPM, procederemos a instalar el framework HapiJs a traves de la ejecucion de los siguientes comandos :

$ sudo npm install --save hapi

En caso de no tener Node instalado descargar el [instalador de Node.js para su sistema operativo](https://nodejs.org/en/download/) y seguir las instrucciones, otra alternativa para instalar Node.js si usa Linux o Mac es usar un manejador de versiones ([nvm](https://github.com/creationix/nvm)) el cual nos da una ventaja de poder usar diferentes versiones de node y ademas no necesitamos permisos de superusario para instalar librerias de node. 

# Instalar nvm

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.30.1/install.sh | bash

# Actualizar el bash\_profile

source ~/.bash\_profile

# Instalar la version estable de node y npm

nvm install stable

# Verificar que node y npm se instalaron correctamente

node --version

npm --version

### Ejemplo 1

Iniciamos un nuevo proyecto y en "entry point" colocamos server.js

$ npm init

Luego de instalar Hapi.Js empezamos a escribir nuestro código para crear nuestro primer servidor, guardamos este código como server.js

var Hapi = require('hapi');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.route({

method: 'GET',

path: '/',

handler: function (request, reply) {

reply('Hola! Amiguito');

}

});

server.start(() => {

console.log('Servidor corriendo :', server.info.uri);

});

Y para ejecutar el programa en el terminal ejecutamos el comando npm start desde la carpeta donde guardamos el archivo

### Ejemplo 2

¿Recuerdan el manejo de Rutas con Node & Express?

Esta ves realizaremos el mismo ejercicio, pero utilizando Node & HapiJs, así que utilizaremos el siguiente código para responder las consultas de forma específica según la ruta que se le pide a través de una llamada al servidor.

Hacemos el backup del archivo server.js anterior y creamos uno nuevo con el siguiente código y lo guardamos nuevamente como server.js

var Hapi = require('hapi');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.route({

method: 'GET',

path: '/{name}',

handler: function (request, reply) {

reply('Hello ' + request.params.name);

}

});

server.start(function () {});

Y para ejecutar el programa en el terminal ejecutamos nuevamente el comando npm start desde la carpeta donde guardamos el archivo

### Ejemplo 3

Sirviendo archivos estáticos

Utilizaremos el plugin Inert para HapiJs que nos permitirá servir archivos estáticos en nuestro servidor.

Hacemos el backup del archivo server.js anterior y creamos uno nuevo con el siguiente código y lo guardamos nuevamente como server.js

var Hapi = require('hapi');

var Inert = require('inert');

var Path = require('path');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.register(Inert, function () {});

server.route({

method: 'GET',

path: '/',

handler: {

file: Path.join(\_\_dirname, 'public/index.html')

}

});

server.start(function () {});

Creamos una carpeta "Public" con el comando mkdir Public donde colocaremos nuestro archivoindex.htmly ademas no olvidemos instalar inert & path.   
Guardamos y ejecutamos el comando npm start

### ****Ejemplos****Directorios, Viewss & Proxy's en HapiJs

### Directories

Imaginemos que en nuestro sitio website necesitamos identificar un área que contenga archivos para hacer testing

Lo primero que se nos ocurre es http://hackspace.la/testing/files/namefile.extension. Donde podemos observar que dentro del root tenemos una carpeta testing y una subcarpeta files que contiene todos los archivos que pasarán por testing.

Esto por el lado del usuario, se ve ordenado, sin embargo, para el área de desarrollo genera symbolic links muy largos. Para optimizar esto, utilizamos Directories para minificar el tamaño del PATH o ruta de archivo.

El usuario puede ver http://hackspace.la/testing/files/namefile.extension

Pero en nuestro servidor solo podemos tener /public/namefile.extension

Creamos un nuevo archivo server.js y utilizamos el siguiente código

var Hapi = require('hapi');

var Inert = require('inert');

var Path = require('path');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.register(Inert, function () {});

server.route({

method: 'GET',

path: '/testing/files/{filename}',

handler: {

directory: {

path: Path.join(\_\_dirname, 'public')

}

}

});

server.start(() =>{

console.log('Servidor corriendo en:', server.info.uri);

});

Creamos un nuevo archivo index.html dentro de la carpeta public y utilizamos el siguiente código o el código que desees. PD: quitar el "#".

<#html>

<#head><#title>Hola Hackspace<#/title><#/head>

<#body>

Run or feed

<#/body>

<#/html>

Y para ejecutar el programa en el terminal ejecutamos nuevamente el comando npm start desde la carpeta donde guardamos el archivo

### Views

Una forma rápida y sencilla de crear un servidor que responda solicitudes a /?name=Key a través de plantillas es utilizando Views con el módulo Vision en HapiJs.   
Instalamos el paquete Vision a través del siguiente comando en la terminal

sudo npm install --save vision

var Hapi = require('hapi');

var Vision = require('vision');

var Path = require('path');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.register(Vision, function () {});

server.views({

engines: {

html: require('handlebars')

},

path: Path.join(\_\_dirname, 'templates')

});

server.route({

method: 'GET',

path: '/',

handler: {

view: 'index.html'

}

});

server.start(() => {

console.log('Servidor corriendo en:', server.info.uri);

});

Creamos un nuevo archivo index.html dentro de la carpeta templates y utilizamos el siguiente código o el código que desees. PD: quitar el "#".

<#html>

<#head><#title>Hola! {{query.name}}<#/title><#/head>

<#body>

Hola! {{query.name}}

<#/body>

<#/html>

Y para ejecutar el programa en el terminal ejecutamos nuevamente el comando npm start desde la carpeta donde guardamos el archivo

### Proxy

Los proxies nos permiten retransmitir solicitudes de un servidor y/o servicio a otro. Esto es muy potente, ya que podemos manejar múltiples servicios como aplicaciones web y construcciones RestAPI.   
Para este caso utilizaremos el paquete H2o2 instalándolo a través de:

sudo npm install --save h2o2

Creamos un archivo server.js e introducimos el siguiente código

var Hapi = require('hapi');

var H2o2 = require('h2o2');

var server = new Hapi.Server();

server.connection({

host: 'localhost',

port: Number(process.argv[2] || 8080)

});

server.register(H2o2, function () {});

server.route({

method: 'GET',

path: '/proxy',

handler: {

proxy: {

host: '127.0.0.1',

port: 65535

}

}

});

server.start(function () {});

Todos los ejemplos han sido tomados y adaptados del Módulo [MakeMeHapi](https://github.com/hapijs/makemehapi)

<https://asciinema.org/>

**Reto 2**Segundo reto del entrenamiento de Backend.

Reto 2 de Backend

En la semana 1, realizaron la creación de 1 API con Express y modelaron los datos con faker.js

Esta segunda semana el reto será consumir los datos de una API

https://developer.valvesoftware.com/wiki/Steam\_Web\_API/Feedback

Tendrán que crear 1 enpoint /{username}/  
que pueda convertir y mostrar los nombres de usuarios de steam en ID's   
No, necesito que hagan todo un frontend para ello, solo quiero que la respuesta sea el ID en una página sin tanto código.

<div>Su ID es {{query.name}}</div>

Por ejemplo mi nombre de usuario en steam es mizhac\_ y mi steam ID es: STEAM\_0:1:42388192

Este es un ejemplo de website que hace esta funcion

www.steamcommunity.com

Y el url que nos permite convertir los nombres de usuarios a ID's es

http://api.steampowered.com/ISteamUser/ResolveNickname/v0001/?nickname=XXXXXXXXX

Suban sus archivos a sus cuentas de Github una vez finalizado el reto, y nos envían el url al formulario.

PD1: No se aceptará recibir el reto de otra forma que no sea el repositorio de github.

PD2: Sí, el API brindado no funciona correctamente, parte de este reto incluye revisar su documentación y hacer los cambios que necesite para que pueda funcionar.