NodeJS is Event Oriented Platform built in Javascript execution time for Chrome. All Node library API’s are asynchronous and unblocked. This means that a NodeJS based server does never wait an API to return data. Instead, the server moves on to the next API and a mechanism of event notification retrieve the response from the previously called API.

* NodeJS is very fast (V8 engine).
* Only one subprocess with loop event.
* Without buffer: NodeJS apps do not store data, instead, they output data by pieces.

In an event-controlled application there is a principal loop which listen the events and then activates a callback function. Callback and events are not the same thing. Callback functions are called when an asynchronous function returns a result, while the event-handler works within the observer frame. Each time an event is triggered, its listener function starts executing.

EventEmitter class is used to bind events with listeners:

Import events module  
**const events = require(‘events’);**

Create an eventEmitterr Object  
**const eventEmitter = new events.EventEmitter();**

Event and event-handler (controller) binding **eventEmitter.on(‘eventName’, eventHandler);**

Throw an event programtically **eventEmitter.emit( ‘eventName’);**

# Event example

// Import events module  
**const events = require('events');**

// Create an event emitter Object  
**const eventEmitter = new events.EventEmiter();**

// Create an event handler  
**const connectHandler = () => {**

**console.log('connection successful');  
}**

// Bind the connection event with the handler  
**eventEmitter.on('connection', connectHandler);**

// Bind the data\_received event with the anonymous function

**eventEmitter.on('data\_received', () => {**

**console.log('data received successfully.');  
});**

// Fire the conncetion event  
**eventEmitter.emit('conncetion');**// Outputs: connection successful. 🡪 data\_received

**console.log('Program Ended');**

# Node installation

* Node.js runtime
  + Performance counters
  + Event tracing (ETW)
* npm package manager
* Online documentation shortcuts
* Add to PATH
  + Node.js and npm
  + npm modules

# Node Lite-Server

## Install Lite-Server

- Launch the Bash command line

- Install lite-server:

**$ npm install -g lite-server**

## New Project

* Create a directory
* Create a Node project:

**$ npm init –y**

package.json:

{

"name": "NodeJS",  
"version": "1.0.0",  
"description": "",  
"main": "index.js",  
"scripts":

{  
"test": "echo \"Error: no test specified\" && exit 1"  
},

"keywords": [],  
"author": "",  
"license": "ISC"

}

* Create an index.html file with the text "Hola Mundo" in the project
* Modify the package.json so in the "start" the lite-server will be launched:

"scripts":  
{  
"start": "lite-server"  
},

## Start and close the server

* Launch the Project:

**$ npm start**

The server is launched and listenning at <http://localhost:3000/>

All plugins introduced at the dependencies JSON of the package.JSON file, are taken from de Node Package Manager (npm). [www.npmjs.com](http://www.npmjs.com)

**$ npm install** 🡪 executes package.JSON and install all plugins writen in the sub-JSON dependencies

[**https://docs.npmjs.com/getting-started/semantic-versioning**](https://docs.npmjs.com/getting-started/semantic-versioning)

<https://blog.risingstack.com/your-first-node-js-http-server/>

<https://jesuslc.com/>

<https://geekytheory.com/node-js-primeros-pasos-y-hola-mundo>

# NodeProjectExample

Creamos la carpeta del nuevo proyecto, entramos en ella y ejecutamos el primer comando:

1. **$ mkdir newProject**
2. **$ cd newProject/**
3. **$ npm init**
4. **$ npm install bootstrap**
5. **$ npm install jquery**
6. **$ npm install Popper.js**

The package.json file has a key called: “dependencies” with the required modules for the project. We can copy the package.json file in a new Project and then execute the command line **$ npm install** 🡪 automatically, the resources will be updated to the last version according to the semantic-versioning description of the dependencies sub-json.

# Node exercises

## Send text/plain

Node uses libraries or modules 🡪 these are objects with its methods and attributes.  
**const httpModule = require('http');**This import the http package with all his methods - listen(), end(), etc - and attributes like url.

**const requestHandler = ( request, response ) => {**

**response.writeHead( 200, {'content-type': 'text-plain'} );**   
// It would be also valid 🡪 text/html

**response.write( 'Hello Node.js Server!' );**

**console.log( request.url );** // console--> /

**respuesta.end();**

**}**

Using the response object (parameter) we can communicate with the client. First to do is setting the HTTP header with the correct confirmation status (200) and with the content-type to send.

**const server = httpModule.createServer( requestHandler );  
const port = 8080;**

**server.listen( port, (err) => {**

**if (err) {**

**return console.log('something bad happened', err)  
}  
console.log( `server is listening on ${port}` )**

**});**

## Send text with HTML code

**const httpModule = require( "http" );**

**const control = ( req, resp ) => {**

**resp.writeHead( 200, {"Content-Type": "text/html"** /\*"charset": "UTF-8", name": "viewport", "content": "width-device-width, initial-scale=1.0", "http-equiv": "X-UA-Compatible", "content": "ie=edge"\*/ **} );**

**resp.write( "<h1>Hola Mundo! \nQué fácil es incluir 'comillas simples' y \"comillas dobles\"</h1>" );**

**resp.end( `  
 <font face = 'Arial'>Mi primer servidor <b>Node<b/></font>  
 ` );  
}**

**const server = httpModule.createServer();** /\*If server is requested, call 'control'\*/   
**server.on( 'request', control );**

**server.listen( 8080 );  
console.log( "server working" );**

## Read file HTML

**const httpModule = require( 'http' );** //Módulo http  
**const fileSystemModule = require( 'fs' );** //Módulo fs

**const message = "\nQué fácil es incluir 'comillas simples' y \" dobles\"";**

**moduloHTTP.createServer( (request, response) => {**

**fileSystemModule.readFile(**

**'./fileSystemNode.html',  
function( error, data ) {** //la ruta ./ es mismo nivel y se podría omitir

//Tiempo real: while read --> write  
 **response.writeHead(**

**200, {'Content-Type': 'text/html'}**

**);**

**response.write('<h1>Typing data from JS file </h1>');  
 response.write( `<h2>${mensaje} (From JS)</h2>`);  
 response.write( data );** //Archivo que lee en la ruta  
 **response.end();**

**}**

**);** //Cierro la función CALLBACK  
**}).listen( 8080 );** //Cierro la función de LÓGICA DEL SERVIDOR  
**console.log(`server is listening on 8080 port`);**

## File System

**const httpModule = require( 'http' );** //Módulo http  
**var urlModule = require( 'url' );** //Módulo url  
**var fileSystemModule = require( 'fs' );** //Módulo fs

**httpModule.createServer( (request, response) => {**

**fileSystemModule.readFile(**

**'../form.html',  
function( error, data ) {** //Tiempo real: while read --> write

**response.writeHead(**

**200, {'Content-Type': 'text/html'}  
);**

**response.write( data );** // the file ridden through the path  **response.end();**

**}  
);** //Cierro la función CALLBACK

**}).listen( 8080 );** //Cierro la función de LÓGICA DEL SERVIDOR

**console.log( "servidor funcionando" );**

## Retrieve URL paths

**const httpModule = require( 'http' );  
const urlModule = require( 'url' );**

**const controlHandler = ( request, response ) => {**

//Determine the requested URL **const page = urlModule.parse( peticion.url ).pathname;**

/\*Envío del código 200 (todo OK) en el encabezado de la página con el tipo MIME text/html para indicar un envío de código HTML al navegador\*/ **response.writeHead( 200, {'content-type': 'text/html'} );**

/\*Preparación del script para enviar al navegador\*/  
 **response.write( '<font face="Arial">' );**

**switch(page) {**

**case '/':  
 response.write( "You are in the landing page" );  
 response.write( "<br/> Add a path route" );  
 break;**

**case '/profile':  
 response.write( "This is your profile account" );  
 break;**

**case '/events':  
 response.write( 'Find all available events' );  
 break;**

**case '/sites':  
 response.write( 'Find all available sites' );**

**}**

**response.write( '</font>' );**

**};**

//Instantiate the server  
**const server = moduloHTTP.createServer( controlHandler );**

// Run the server  
**const port = 8080;**

**server.listen( port, (err) => {**

**if (err) {**

**return console.log('something bad happened', err);  
}**

**console.log(`server is listening on ${puerto}`);**

**});** <https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP>

<https://www.joezimjs.com/javascript/3-ways-to-parse-a-query-string-in-a-url/>

YEOMAN 🡪 scaffolding

Comando: **yo**

Comprar un dominio, configurar las DNS para que apunten al router de la máquina servidor y permitir la entrada por el puerto 80, además de establecer un IP fija.