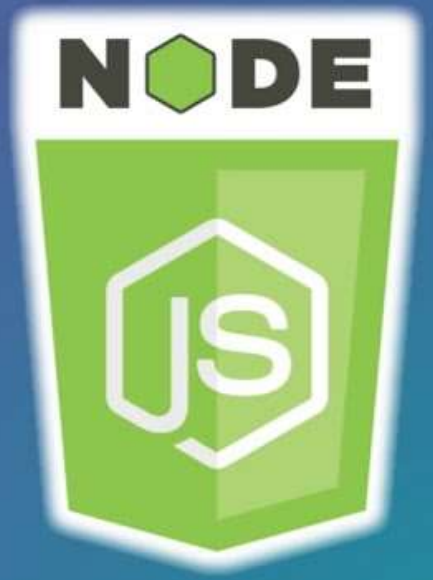
Using pure Node.Js

NodeJs Develop Web App (HTTP Server, HTTP GET, HTTP POST)



Fernando Saez

HTTP

HTTP (HyperText Transfer Procolol) o (Protocolo de transferencia de hipertexto).

Permite la transferencia de datos entre un cliente (Navegador) y un servidor web.

protocolo://host[:puerto][/ruta y archivo][?parámetros]

https://www.google.com/search?q=perro

Funcionamiento de HTTP

- El protocolo es iniciado por el cliente con un "request" que es casi siempre contestado por el servidor con un "response".
- Un request genérico tiene la forma:

METODO URI PROTOCOLO CrLf
HEADERS* CrLf
Datos

METODO: GET o POST
URI: http://unserver/un_recurso
PROTOCOLO: HTTP/1.1
CrLf: Retorno de carro + nueva línea.

Ej: GET http://www.yahoo.com HTTP/1.1

```
User-Agent: Mozilla/5.0 (Windows
NT 5.1) Gecko/20060909
Firefox/1.5.0.7
Accept: text/html, image/png, */*
Accept-Language: en-us,en;q=0.5
Cookie: rememberme=true;
PREF=ID=21039ab4bbc49153:FF=4
```

Datos…

Funcionamiento de HTTP

- El server responde con un “response” de la forma:

PROTOCOLO STATUS VALOR CrLf
HEADERS*
Content-Type: TIPO CrLf
Datos

HTTP/1.1 200 OK
Date: Mon, 12 Jun 2000 14:04:28 GMT
Server: Apache/1.3.9 (Unix)
ApacheJServ/1.1
Connection: close
Content-Type: text/html
Datos…

Funcionamiento de HTTP

```
<HTML>
<HEAD>
<TITLE>Ejemplo</TITLE>
</HEAD>
<BODY>
Hola esta es una prueba
<IMG SRC="prueba.gif">
</BODY>
</HTML>
```

BROWSER	SERVER
GET http://www.prueba.com/index.html HTTP / 1.1	HTTP/1.1 200 OK Date: Tue, 13 Jun 2000 14:15:45 GMT Server: Apache/1.3.9 (Unix) PHP/4.0.0 Last-Modified: Tue, 13 Jun 2000 14:09:05 GMT ETag: "5804d-73-39464081" Accept-Ranges: bytes Content-Length: 115 Connection: close Content-Type: text/html <HTML> <HEAD> <TITLE>Ejemplo</TITLE> </HEAD> <BODY> Hola esta es una prueba <IMG SRC="prueba.gif"> </BODY> </HTML>
GET http://www.prueba.com/prueba.gif HTTP / 1.1	HTTP/1.1 200 OK Date: Tue, 13 Jun 2000 14:18:22 GMT Server: Apache/1.3.9 (Unix) PHP/4.0.0 Last-Modified: Tue, 13 Jun 2000 14:07:36 GMT ETag: "5804e-2b2-39464028" Accept-Ranges: bytes Content-Length: 690 Connection: close Content-Type: image/gif GIF89aGÔÿ11B99ZRJkcR-œk½Î ½„Æµ{µ„„{ZÔÆŒï ÞœçÔ"-{sRskJ„„{RkcBœ"k¥{sJµ (CORTADO)

Status de Response

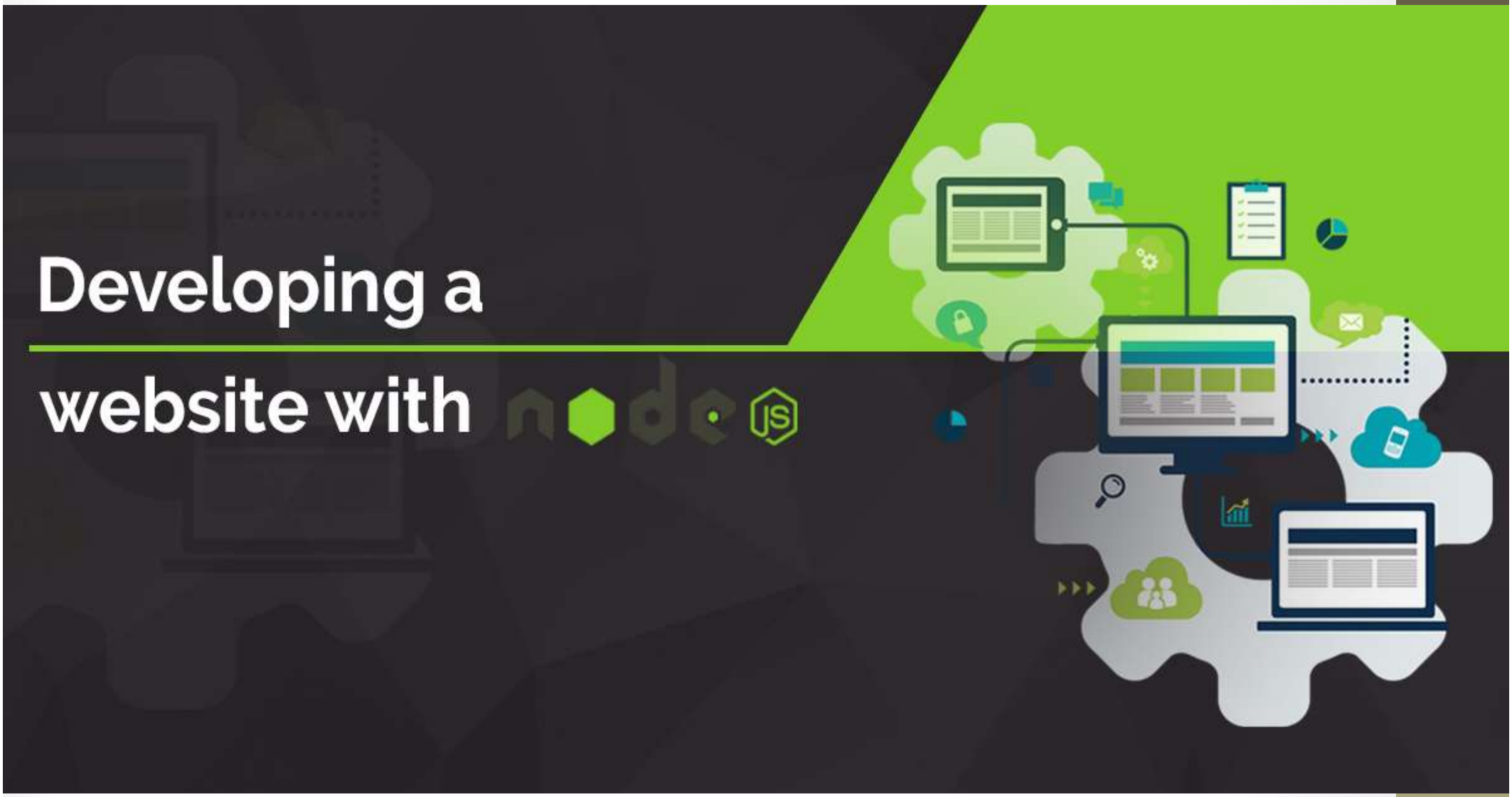
- El primer dígito del código de estado define la clase de Response. Los últimos 2 dígitos no tienen categorización.
  - 1xx: Informacional – Request recivido, proceso continua.
  - 2xx: Exitoso – La acción fue recivida exitosamente, entendida y aceptada
  - 3xx: Redirección – Debe realizar una acción, en orden a completar el request
  - 4xx: Client Error – El request contiene error en la sintaxis o no puede ser procesado.
  - 5xx: Server Error – El server fallo al procesar un request válido.

Status de Response

```
| "101"  ; Section 10.1.2: Switching Protocols
| "200"  ; Section 10.2.1: OK
| "201"  ; Section 10.2.2: Created
| "202"  ; Section 10.2.3: Accepted
| "203"  ; Section 10.2.4: Non-Authoritative Information
| "204"  ; Section 10.2.5: No Content
| "205"  ; Section 10.2.6: Reset Content
| "206"  ; Section 10.2.7: Partial Content
| "300"  ; Section 10.3.1: Multiple Choices
| "301"  ; Section 10.3.2: Moved Permanently
| "302"  ; Section 10.3.3: Found
| "303"  ; Section 10.3.4: See Other
| "304"  ; Section 10.3.5: Not Modified
| "305"  ; Section 10.3.6: Use Proxy
| "307"  ; Section 10.3.8: Temporary Redirect
| "400"  ; Section 10.4.1: Bad Request
| "401"  ; Section 10.4.2: Unauthorized
| "402"  ; Section 10.4.3: Payment Required
| "403"  ; Section 10.4.4: Forbidden
| "404"  ; Section 10.4.5: Not Found
| "405"  ; Section 10.4.6: Method Not Allowed
| "406"  ; Section 10.4.7: Not Acceptable

| "407"  ; Section 10.4.8: Proxy Authentication Required
| "408"  ; Section 10.4.9: Request Time-out
| "409"  ; Section 10.4.10: Conflict
| "410"  ; Section 10.4.11: Gone
| "411"  ; Section 10.4.12: Length Required
| "412"  ; Section 10.4.13: Precondition Failed
| "413"  ; Section 10.4.14: Request Entity Too Large
| "414"  ; Section 10.4.15: Request-URI Too Large
| "415"  ; Section 10.4.16: Unsupported Media Type
| "416"  ; Section 10.4.17: Requested range not satisfiable
| "417"  ; Section 10.4.18: Expectation Failed
```

https://developer.mozilla.org/es/docs/Web/HTTP/Status



Algunas aclaraciones!!

¿Como iniciar un Proyecto?

Requisitos: Tener Node instalado

1.- Crear un directorio que funcione como contenedor del Proyecto. > mkdir appnode

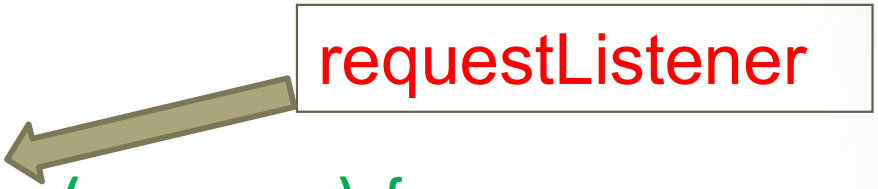
2.- > npm init

3.- Crear la aplicación: Crear index.js y editarlo.

4.- Ejecutar la aplicación > node index.js

App Hello World

- Crear el archive index.js con el siguiente código:




```
var http = require('http');
http.createServer(function (req, res) {
        res.writeHead(200, {'Content-
        Type': 'text/plain'});
        res.write('Hello World!');
        res.end();
}).listen(8081);
```

// Consola mostrara el siguiente mensaje

console.log('Server running at http://127.0.0.1:8081/');

Servidor Web con Node.js

```
const http=require('http');
const servidor=http.createServer((req,res) => {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(`<!doctype html><html><head></head>
                  <body><h1>Sitio en desarrollo
</h1></body></html>`);
  res.end();
});

servidor.listen(8888);
console.log('Servidor web iniciado');
```

Datos que envia el navegador

¿Que sucede cuando el navegador solicita un recurso y envia parámetros?

http://localhost:8888/carpeta1/pagina1.html?param1=10&param2=20

```
const http=require('http');
const {URL}=require('url');
const servidor=http.createServer((req,res) => {
  const baseURL = "http://" + req.headers.host;
  const objUrl = new URL(req.url, baseURL);
  console.log('camino completo del recurso y parametros: '+objUrl.href);
  console.log('solo el camino y nombre del recurso     :'+ objUrl.pathname)
  console.log('parametros del recurso                  :'+ objUrl.searchParams)
  res.writeHead(200, {'Content-Type': 'text/html'});
....
```

Retornar páginas estáticas

```
const servidor=http.createServer( (pedido,respuesta) => {
1  const objUrl = new URL(req.url, "http://" + req.headers.host;);
2  let pathnameStatic ='static'+objetourl.pathname;
3  if (pathnameStatic =='static/')
       pathnameStatic ='static/index.html';
4  fs.stat(camino, error => {  //stat() determina si existe el archive pedido
   if (!error) {
5    fs.readFile(camino, (error,contenido) => {
       if (error) {
6        respuesta.writeHead(500, {'Content-Type': 'text/plain'});
         respuesta.write('Error interno');
         respuesta.end();
       } else {
7        respuesta.writeHead(200, {'Content-Type': 'text/html'});
         respuesta.write(contenido);
```

else

Retornar recursos estáticos

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

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

Tipos MIME

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

Cache de recursos estáticos

```
const cache={};

if (cache[camino]) {
    const vec = camino.split('.');
    const extension=vec[vec.length-1];
    const mimearchivo=mime[extension];
    respuesta.writeHead(200, {'Content-Type': mimearchivo});
    respuesta.write(cache[camino]);
    respuesta.end();
    console.log('Recurso recuperado del cache:'+camino);
else {…}
```

Routing

Routing: define la forma en que los end-points de la aplicación manejan las solicitudes del cliente.

End-point: unidades de código ejecutables que manejan las solicitudes de la aplicación.

Implementación del enrutamiento en Node.js

- Mediante el uso de Framework
- Sin usar Framework

Routing sin framework

```
http.createServer(function (req, res) { // ….
res.writeHead(200, {'Content-Type': 'text/html'});
var url = req.url;
if(url ==='/about') {
    res.write(' Welcome to about us page');
    res.end();
 } else if(url ==='/contact') {
    res.write(' Welcome to contact us page');
    res.end();
 } else {
    res.write('Hello World!');
    res.end();
 }
```

Modularizando nuestra App (1)

Modularizamos en 3 archivos:

1. router.js
2. HandlerRequest.js
3. server.js
4. index.js

Preliminar

```
//router.js
function route() {
   if(url ==='/about') {
    res.write(' Welcome to about us
page');
    res.end();
 } else if(url ==='/contact') {
    res.write(' Welcome to contact us
page');
    res.end();
 } else {
    res.write('Hello World!');
    res.end();
 }
}
exports.route = route;
```

Modularizando nuestra App (1.1)

```
//router.js
function route(handle, pathname, response, request) {

if (typeof handle[pathname] === 'function') {
handle[pathname](response, request);
} else {
console.log("No request handler found for " + pathname);
response.writeHead(404, {"Content-Type": "text/html"});
response.write("404 Not found");
response.end();
}
}
exports.route = route;
```

1

Definitivo!!

Modularizando nuestra App (2)

```
//handlerRequest.js

function index(response, request) {
response.writeHead(200, {"Content-Type": "text/html"});
response.write('<h1> Hello World </h1>');
response.end();
}

function about(response, request) {
response.writeHead(200, {"Content-Type": "text/html"});
response.write('<h1> Welcome to about us page </h1>');
response.end();
}
…
exports.index = index;
exports.about = about;
exports.contact = contact;
```

Modularizando nuestra App (3)

```
//server.js

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

1 function start(route, handle) {
    function requestListener(request, response) {
        Const {pathname}=new URL(req.url, "http://" + req.headers.host;);
        console.log("Request para ruta " + pathname + " recibido.");
    3 route(handle, pathname, response, request);
    }

    http.createServer(requestListener).listen(8888);
    console.log("Servidor iniciado.");
2 }
exports.start = start;
```

Modularizando nuestra App (4)

```
//index.js
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}

handle["/"] = requestHandlers.index;
handle["/about"] = requestHandlers.about;
handle["/contact"] = requestHandlers.contact;

server.start(router.route, handle);
```