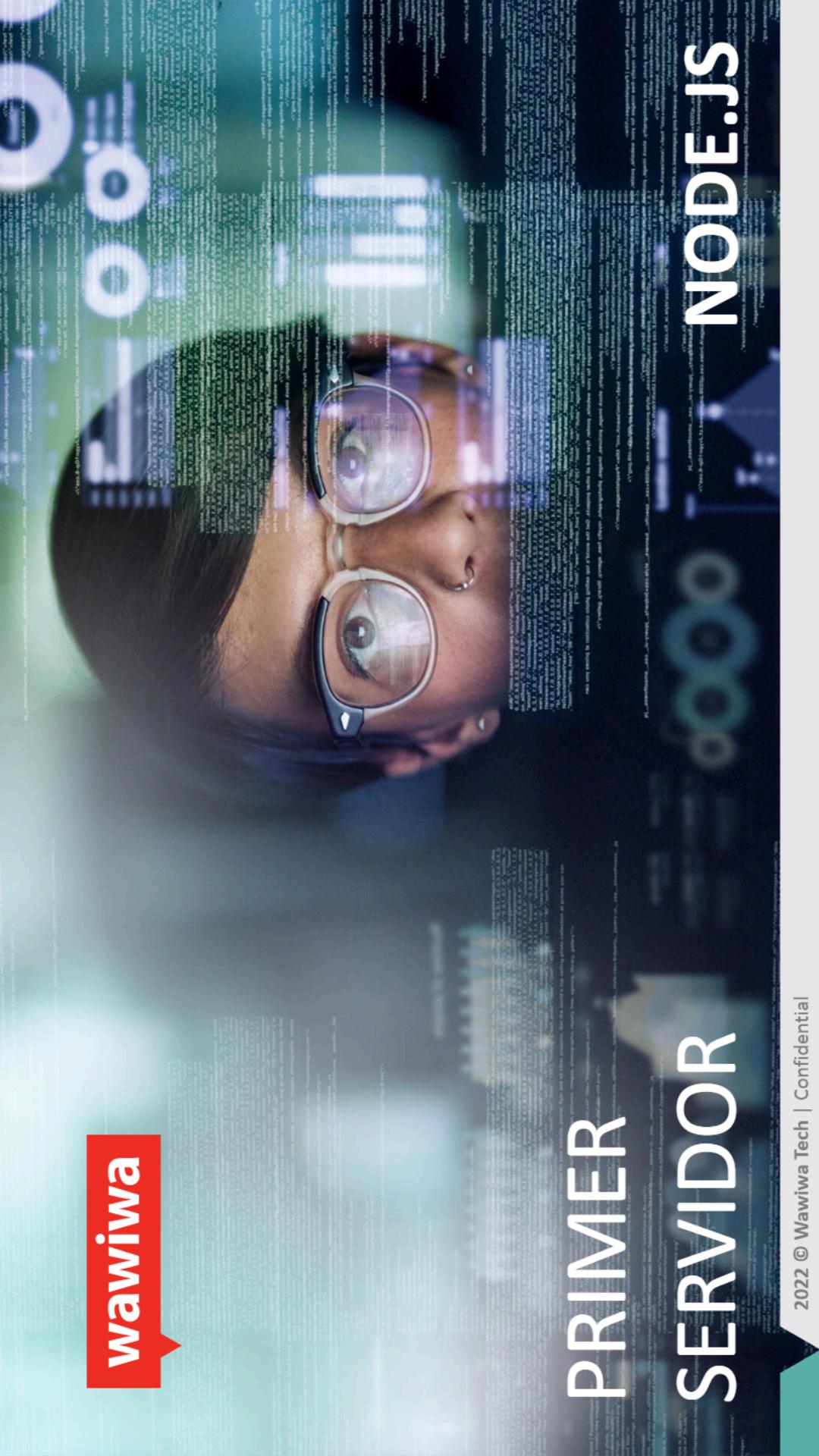




wawiwa

PRIMER SERVIDOR

NODE.JS



wawiwa

<meta name="description" content="Buy royalty-free images for your website, poster, afford!"/>
<meta name="keywords" content=" stock photo, stock photography, stock image, picture, images, & more images</title>
use images for your website, poster,                      <img alt="apple-touch-icon-precomposed.png" data-bbox="145

Directorio de archivos

En este capítulo tienes 8 archivos fuente en la carpeta del capítulo.

server1.js

server2.js

server3.js

server4.js

server5.js

server6.js

server7.js

red/red1.js

red/red2.js

red/red3.js

¿Cómo funciona el internet?

Cuando ingresamos una dirección recibimos una página web que contiene texto, imágenes y más.



¿Cómo funciona el internet?

Cada dirección en el internet tiene una dirección de **ip**.

La dirección de ip es número que es usado para identificar clientes y servidores y es asignados dinámicamente dentro de la red local.

Pueden haber dos direcciones de ip iguales, si es que no están conectadas a la misma red local.

Es por eso que cada computadora también debe tener una dirección física que no puede ser cambiada, llamada dirección **MAC**.

¿Cómo funciona el internet?

Para encontrar el ip de un servidor, abre el cmd:

- **Presiona la tecla win + r e ingre 'cmd'**
- Corre el comando 'ping':

```
C:\Users\x5-i7>ping google.co.il  
Pinging google.co.il [216.58.206.3] with 32 bytes of data:  
Reply from 216.58.206.3: bytes=32 time=70ms TTL=112
```

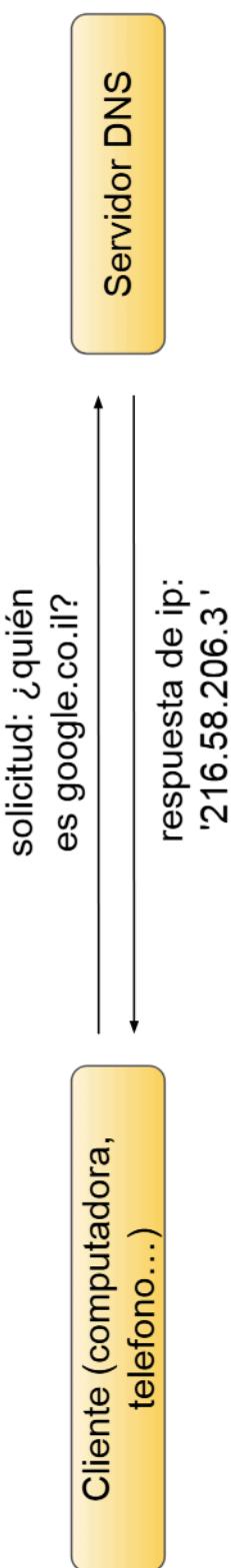
Observa que puedes mandarte información a ti mismo con la ip '127.0.0.1'.

¿Cómo funciona el internet?

DNS:

Domain Name System (sistema de nombres de dominio) es un protocolo para acceder a bases de datos, para permitir a los clientes traducir direcciones (urls) legibles para humanos en nombres de dominios (ips)

Más sobre dns : https://en.wikipedia.org/wiki/Domain_Name_System



¿Cómo funciona el internet?

Puerto:

Un proceso específico por el que los programas pueden transmitir información directamente.

Digamos que IP es la dirección de un edificio. Para mandar una carta debemos saber la dirección (IP) y el departamento (puerto) también..

¿Cómo funciona el internet?

Socket (enchufe):

Una red de sockets es un cliente para transmitir información entre dos procesos en la red.

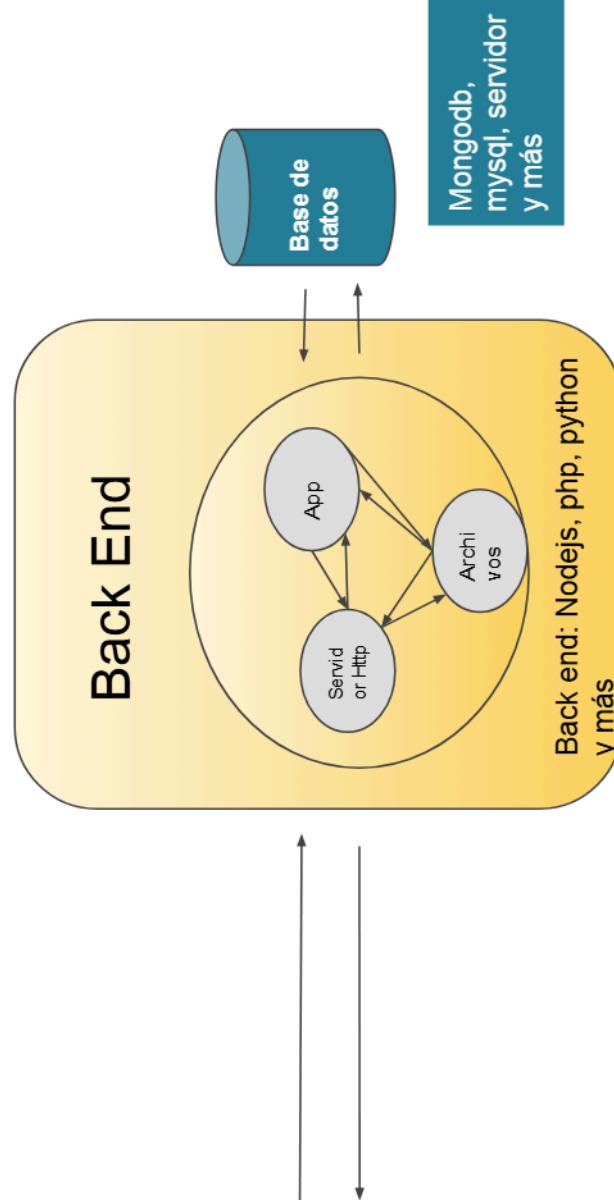
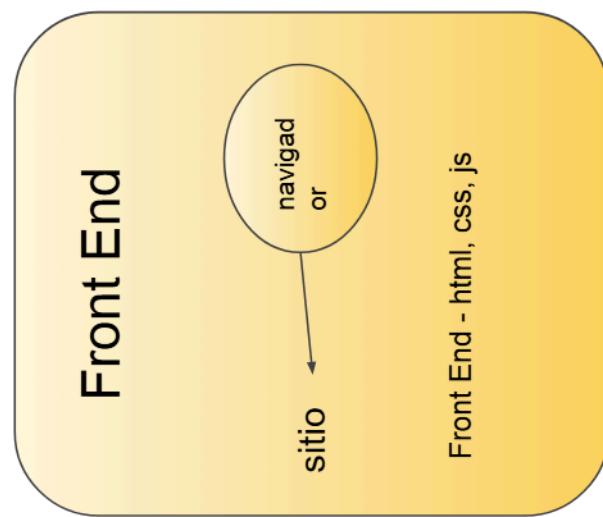
Una dirección de socket es la combinación de ip y puerto.

Un cliente que manda información a 1.2.3.4:1737

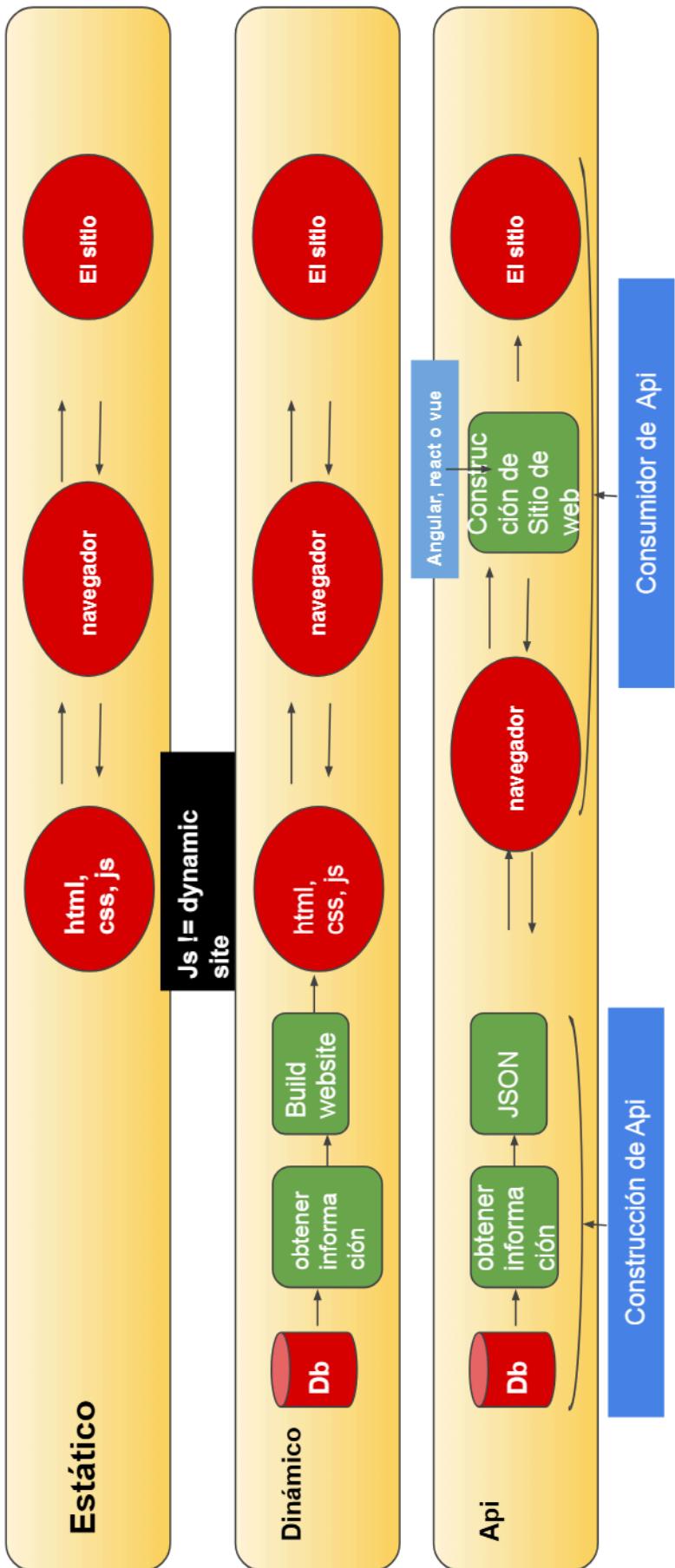
Transmisión de información

Un servidor con ip 1.2.3.4 que escucha en el puerto 1737

Front-End vs. Back-End



Estático, dinámico y Api



Api

Api



navegad
ores



Android
app



ios app

Robots y
iot

Y más

Primer servidor

En este capítulo construiremos el Primer servidor en esta guía, con módulo http.

En un nuevo archivo llamado "server1.js":

```
const http = require('http');
res.write('Hello World!');
res.end();
}
```

Usa el módulo http, que está incorporado en node

El servidor responde con "Hello World" sin importar cuales es la solicitud

Creamos un servidor usando el módulo http y pasandole server_function

El servidor va a escuchar en el puerto 3000

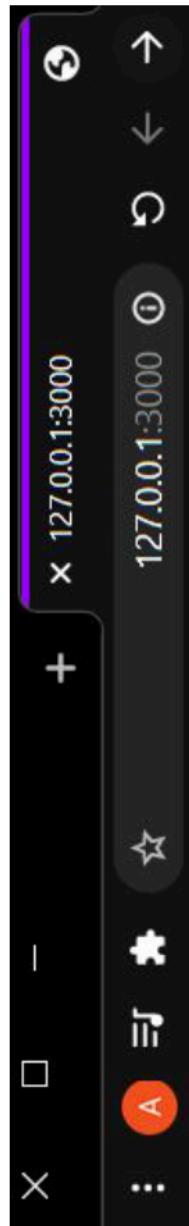
Primer servidor

Cómo creamos el archivo server1.js bajo un directorio, necesitamos cambiar el nombre del directorio para correr el programa:

'cd' significa "Change Directory"

```
chap12-mongodb % cd '/Users/asafamir/Desktop/my-node-proj/chap3-firstserver'  
chap3-firstserver % node server1
```

Ahora abre el navegador y solicita nuestra computadora (127.0.0.1, o localhost) con puerto 3000:



Primer servidor

Para entrenar, construyamos otro servidor, en un nuevo archivo llamado 'server2.js':

```
const http = require('http');

http.createServer((req, res) => res.end('Hello World 2!')).listen(3001);
```

En la terminal presiona **ctrl + c** para salir del programa previo (server1) y correr server 2.

Ahora abre un navegador y solicita en tu computadora con puerto 3001:



Primer servidor

Y el último servidor de este capítulo 'server3.js':

```
const http = require('http');

http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html')
  .end("client asked " + req.url);
}).listen(3000);
```

La respuesta puede tener un "header", que contiene más información del servidor.

En este header, '200' indica el código del estatus, que significa 'OK' (la información que el usuario solicitó fue encontrada).

Y el tipo de contenido de la respuesta 'text/html', que puede ser css o js o lo que queramos.



Podemos ver que el usuario solicitó ver los datos en el archivo 'b' que en el directorio 'a' del servidor.

Hazlo tú mismo 1

Crea un servidor que responda a 127.0.0.1:3000 con tu nombre y tu apellido.

Hazlo tú mismo 1 - solución

```
const http = require('http');

http.createServer( (req, res) => {
    res.setHeader('Content-Type', 'text/html')
    .end("My name is Mike Lewis");
}) .listen(3000);
```

Estructura Url

https://cnn.com/somepath/?product=ball&price=50

protocolo

dominio

ruta

Query string
(cadena de consulta)

Primer Servidor - Enrutamiento

Y el último servidor en este capítulo 'server4.js':

```
const http = require('http');

http.createServer(({ url: path }, res) => {

  if (path === "/")
    res.write("client asked for home page");

  else if (path === "/product")
    res.write("client asked for products");

  else if (path === "/persons")
    res.write("client asked for persons");

  res.setHeader('Content-Type', 'text/html').end();

}).listen(3000);
```

client asked for home page

client asked for products

client asked for persons

← → C ① localhost:3000

← → C ① localhost:3000/product

← → C ① localhost:3000/persons

Primer servidor - Enrutamiento

Y el último servidor en este capítulo 'server4.js':

```
const http = require('http');

http.createServer(({ url, path }, res) => {
  if (path === '/') {
    res.write("client asked for home page");
  } else if (path === "/product") {
    res.write("client asked for products");
  } else if (path === "/persons") {
    res.write("client asked for persons");
  } else {
    res.write("page not found");
    res.writeHead(404);
  }
})

res.setHeader('Content-Type', 'text/html').end();
}).listen(3000);
```

The screenshot shows the Network tab in the Chrome DevTools. A single request is listed:

Name	Status	Type	Initiator	Size	Time	Waterfall
index	404	document	Other	155 B	1 ms	

Below the table, the Headers and Response sections are visible, showing the response content 'page not found'.

Primer servidor - Enrutamiento

Y el último servidor en este capítulo 'server5.js':

```
const http = require('http');

http.createServer(({ url, path }, res) => {
  if (path === "/")
    res.write("client asked for home page");
  else if (path === "/product")
    res.write("client asked for products");
  else if (path === "/persons")
    res.write("client asked for persons");
  else {
    res.statusCode = 404;
    res.write("<h1>page not found</h1>");
  }
  res.setHeader('Content-Type', 'text/html').end();
}).listen(3000);
```

Hazlo tú mismo 2

Construye un servidor en el puerto 3000 que responda al usuario de la siguiente manera:

1. Para la url '**/firstname**' - retorna "Miki"
2. Para la url '**/lastname**' retorna "Mimi"
3. Para cualquier otra solicitud responde con "**I don't know how to respond to that**"

```
← → C ⓘ localhost:3000  
I dont know how to respond to that
```

```
← → C ⓘ localhost:3000/lastname  
Mimi
```

```
← → C ⓘ localhost:3000/firstname  
Miki
```

Hazlo tú mismo 2 - solución

```
const http = require('http');

http.createServer({ url: path }, res) => {
  if (path === '/firstname')
    res.write("Miki");
  else if (path === "/lastname")
    res.write("Mimi");
  else
    res.write("I don't know how to respond to that");

  res.setHeader('Content-Type', 'text/html').end();
} ).listen(3000);
```

SearchParams - server6.js:

```
const http = require('http');
const url = require('url');

http.createServer((req, res) => {
  const baseURL = req.protocol + '://' + req.headers.host + '/';
  const reqURL = new URL(req.url, baseURL);
  const searchParams = new URLSearchParams(reqURL.searchParams);

  searchParams.has('firstname') === true; // true
  searchParams.get('firstname') === "mike"; // true
  searchParams.getAll('age'); // []
  searchParams.get('foo') === null; // true
  searchParams.toString(); // firstname=mike&lastname=even
  searchParams.set('age', '12');
  searchParams.toString(); // firstname=mike&lastname=even&age=12
  searchParams.delete('firstname');
  searchParams.toString(); // lastname=even&age=12

  res.setHeader('Content-Type', 'text/html')
  .end("<h1>page not found</h1>");
}) .listen(3000);
```

← → ⌂ ⓘ localhost:3000/persons?firstname=mike&lastname="even"

searchParams

SearchParams

```
URL {  
  href: 'undefined://localhost:3000/persons?firstname=mike&lastname=%22even%22',  
  origin: 'null',  
  protocol: 'undefined:',  
  username: '',  
  password: '',  
  host: 'localhost:3000',  
  hostname: 'localhost',  
  port: '3000',  
  pathname: '/persons',  
  search: '?firstname=mike&lastname=%22even%22',  
  searchParams: URLSearchParams { 'firstname' => 'mike', 'lastname' => '"even"' }  
  hash: ''  
}  
  
URLSearchParams { 'firstname' => 'mike', 'lastname' => '"even"' }  
[  
  ['firstname', 'mike'],  
  ['lastname', '"even"'],  
]  
true  
true  
[]  
true  
firstname=mike&lastname=%22even%22  
undefined  
firstname=mike&lastname=%22even%22&age=12  
lastname=%22even%22&age=12  
□
```

SearchParams - itera en los parámetros de búsqueda- server7.js:

```
const http = require('http');
const url = require('url');

http.createServer((req, res) => {

  const baseURL = req.protocol + '://' + req.headers.host + '/';
  const reqUrl = new URL(req.url, baseURL);

  const searchParams = new URLSearchParams(reqUrl.searchParams);

  // Iterate the search parameters.
  for (const p of searchParams) console.log(p);
}) .listen(3000);
```

// Iterar los parámetros de búsqueda.

```
for (const p of searchParams) console.log(p);
}) .listen(3000);
```

search parameters

localhost:3000/persons?firstname=mike&lastname=even

```
[ 'firstname', 'mike' ]
[ 'lastname', 'even' ]
```

Hazlo tú mismo 3

Construye un servidor en el puerto 3000 que responda al usuario de la siguiente manera:

1. Para la url '**/params**' - retorna todos los parámetros
2. Para cualquier otra solicitud responda con "**I don't know how to respond to that**"

```
← → C ⓘ localhost:3000/test  
a=5&b=6  
I dont know how to respond to that
```

Hazlo tú mismo 3 solución

```
const http = require('http');

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  const baseURL = req.protocol + '://' + req.headers.host + '/';
  const reqUrl = new URL(req.url, baseURL);
  const path = reqUrl.pathname;

  if (path === "/params") {
    const searchParams =
      new URLSearchParams(reqUrl.searchParams);
    res.write(searchParams.toString());
  }
  else {
    res.write("I don't know how to respond to that");
  }
  res.end();
}) .listen(3000);
```

Hazlo tú mismo 1

Crea un servidor que responda a 127.0.0.1:3000 con el nombre de tu ciudad y el nombre de tu país.

Hazlo tú mismo 2

Construye un servidor en el puerto 3000 que responda al usuario de la siguiente manera:

1. Para la url '`/city`' - retorna "Paris"
2. Para la url '`/country`' retorna "Romania"
3. Para cualquiera otra solicitud responde con "**I don't know how to respond to that**"

Hazlo tú mismo 3

Construye un servidor en el puerto 3000 que responda al usuario de la siguiente manera:

El servidor va a recibir dos parámetros a y b

1. Para la url **/plus?a=5&b=6** - retorna la suma de los parámetros
2. Para la url **/mult?a=5&b=6** - retorna los parámetros multiplicados
3. Para otra solicitud responde con "**I don't know how to respond to that**"

<pre>← → C localhost:3000/plus?a=5&b=6</pre>	<pre>← → C localhost:3000/mult?a=5&b=6</pre>	<pre>← → C localhost:3000/mult?a=5</pre>
11	30	I dont know how to respond to that



wawiwa

¿Preguntas?