



wawiwa

CONSTRUIR UN SERVIDOR SIN EXPRESS

NODE.JS

Recursos para este capítulo

Carpeta - chap6-server-without-express

Leer el contenido de un archivo

chap6-server-without-express/part1-teacher-demo/index1.js

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

  if (path === '/') res.end("client asked for home page");
  else if (path === '/product') res.end("client asked for products");
  else if (path === '/persons') res.end("client asked for persons");
  else if (path === '/api') {
    fs.readFile(`.${__dirname}/data/data.json`, 'utf-8', (err, data) => {
      console.log(JSON.parse(data));
      res.setHeader('Content-Type', 'application/json').end(data);
    });
  }
  else {
    res.writeHead(404, { 'Content-Type': 'text/html' })
    .end("<h1>Page not found</h1>");
  }
}).listen(3000);

} ← → ⌂ ⓘ localhost:3000/api
```

The code defines a simple HTTP server using Node.js's built-in `http` module. It handles requests for the root path ('/'), product pages ('/product'), person pages ('/persons'), and API endpoints ('/api'). For API requests, it reads a JSON file named 'data.json' located in the same directory as the script. The server uses the `Content-Type` header to indicate the response is JSON. It also includes a 404 error handling block for other paths.

```
[ { "title": "Yellow Ball",
  "description": "This is yellow ●",
  "image": "●" } ]
```

The JSON data in 'data.json' is shown as an array of objects. Each object has three properties: 'title' (containing 'Yellow Ball'), 'description' (containing 'This is yellow ●'), and 'image' (containing a yellow circle emoji).

Ler el contenido de un archivo

chap6-server-without-express/part1-teacher-demo/index2.js

Ler el contenido una sola vez

```
const pageData =  
  fs.readFileSync(`.${__dirname}/data/data.json`, 'utf-8');  
  
console.log(JSON.parse(pageData));  
  
http.createServer((req, res) => {  
  const baseURL = req.protocol + '://' + req.headers.host + '/';  
  const path = (new URL(req.url, baseURL)) . pathname;  
  
  if (path === "/")  
    res.end("client asked for home page");  
  else if (path === "/product")  
    res.end("client asked for products");  
  else if (path === "/persons")  
    res.end("client asked for persons");  
  else if (path === "/api")  
    res.setHeader('Content-Type', 'application/json')  
    .end(products);  
  else  
    res.writeHead(404, { 'Content-Type': 'text/html' })  
    .end("<h1>Page not found</h1>");  
}); listen(3000);
```

```
[  
  {  
    title: 'Yellow Ball',  
    description: 'This is yellow ball',  
    image: 'yellow_ball.png'  
]
```

Archivo CSS

chap6-server-without-express/part1-demo/templates/page.html

```
* {  
    margin: 0;  
    padding: 0;  
    box-sizing: inherit;  
}  
  
html { box-sizing: border-box; }  
  
body { background:tomato; }  
.container { padding:20px; }  
.cards { padding: 5px; }  
.card {  
    border: 5px solid royalblue;  
    width: 25%;  
    border-radius: 10px;  
    padding: 5px;  
    font-size: 20px;  
    margin:10px 10px;  
    float: left;  
}
```

Archivo Html

chap6-server-without-express/part1-demo/templates/page.html

Puedes usar cualquier símbolo.
(No es obligatorio usar el símbolo
%)

```
<div class="container">
  <h1> %TITLE% </h1>
  <div>
    <p>Description : { %DESCRIPTION% } </p>
    <p>Image : { %IMAGE% } </p>
  </div>
</div>
```

🔊 { %TITLE% } 🐶
Description : { %DESCRIPTION% }
Image : { %IMAGE% }

Función replaceTemplate

```
function replaceTemplate(template, data) {  
    return template.replace(/ {\%TITLE\%} /g, data.title)  
        .replace(/ {\%DESCRIPTION\%} /g, data.description)  
        .replace(/ {\%IMAGE\%} /g, data.image);  
}
```

"Template" es el contenido de un archivo html

Data es el contenido del archivo data.json

Vamos a reemplazar el TITLE, DESCRIPTION Y IMAGE con el contenido del archivo data[0].title, data[0].description, data[0].image

Leer contenido de html

```
const templatePage =  
  fs.readFileSync(`${__dirname}/templates/page.html`, 'utf-8');
```

Leer el contenido de un archivo

chap6-server-without-express/part1-demo/templates/index3.html

Lee el contenido una sola vez

```
const pageData = fs.readFileSync(`.${__dirname}/data/data.json` , 'utf-8') ;  
const pageobjData = JSON.parse(pageData) ;  
  
function replaceTemplate(template, data) {  
    return template.replace(/{\%TITLE%}/g, data.title)  
        .replace(/{\%DESCRIPTION%}/g, data.description)  
        .replace(/{\%IMAGE%}/g, data.image) ;  
}  
  
const templatePage = fs.readFileSync(`.${__dirname}/templates/page.html` , 'utf-8') ;  
  
http.createServer((req, res) => {  
    const baseURL = req.protocol + '://' + req.headers.host + '/';  
    const path = (new URL(req.url, baseURL)) .pathname;  
  
    if (path === "/") {  
        res.setHeader('Content-Type', 'text/html')  
        .end(replaceTemplate(templatePage, pageobjData[0])) ;  
    }  
    else {  
        res.writeHead(404, {'Content-Type': 'text/html'})  
        .end("<h1>Page not found</h1>");  
    }  
}).listen(3000) ;
```

Reemplaza el contenido de un archivo html con el archivo json

Leer el contenido de un archivo

← → ⌂ ⓘ localhost:3000



Yellow Ball

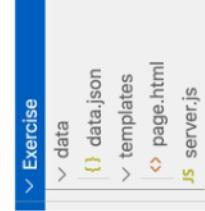
Description : This is yellow

Image :



Hazlo tú mismo 1

Agrega esto a la carpeta
“data”



1. Crea una carpeta con 2 directorios

- a. Información que contiene data.json

```
[  
  {  
    "product": "Heart",  
    "Description": "Pink Heart",  
    "price": "50",  
    "image": "heart"  
  }  
]
```

- b. templates - contiene page.html - Contenido de las siguientes diapositivas de page.html..

1. Agrega a la carpeta llamada server.js

page.html

```
body { background: hsl(110, 100%, 64%); }
.container { padding: 20px; }
.cards { padding: 5px; }
.card {
    border: 5px solid royalblue;
    width: 25%;
    border-radius: 10px;
    padding: 5px;
    font-size: 20px;
    margin: 10px;
    float: left;
}
```

Agrega este archivo a la carpeta "templates"



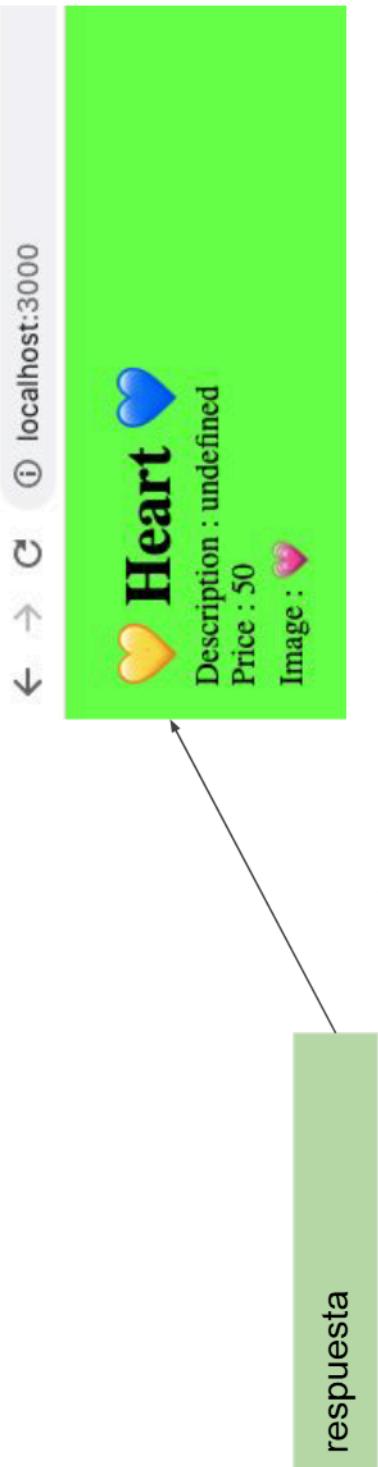
page.html - continuación

```
<div class="container">
  <h1> { %PRODUCT% } </h1>
  <div>
    <p>Description : { %DESCRIPTION% } </p>
    <p>Price : { %PRICE% } </p>
    <p>Image : { %IMAGE% } </p>
  </div>
</div>
```



Continuación del ejercicio

Crea un servidor que escuche en el puerto 3000 y responda con el data.json relevante.



Construye un Servidor sin Express - bueno saber

En todos los ejemplos que vimos, el usuario recibió la misma respuesta para cada url que ingresó.

Construyamos un servidor donde controlemos las páginas que el usuario reciba.

Primero, bajo el directorio 'chap 4' crea un directorio llamado 'public' y dentro de él un archivo llamado 'index.html' y dale algo de contenido HTML:

Construye un Servidor sin Express

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>My Server</title>
  </head>

  <body>
    <h1>This is my server!</h1>
    And here is some more content
  </body>
</html>
```

Construye un Servidor sin Express

Y bajo 'chap6' crea un archivo llamado 'server-only-index.js':

```
http.createServer((req, res) => {
  console.log(` ${req.method} request for ${req.url}`);
  if (req.url === "/") {
    fs.readFile("./public/index.html", "UTF-8", (err, html) =>
      res.setHeader("Content-Type": "text/html").end(html));
    return;
  }
  res.writeHead(404, { "Content-Type": "text/plain" })
  .end("404 file not found");
}) .listen(3000);
```

req.method indica el método REST el usuario usó para acceder a la información (GET, POST...)

Para la url '/', que es por defecto, responde con index.html'. De otra manera responde con un código de estado 404, que significa 'page not found' (página no encontrada).

De esta manera también podemos responder diferente a cada url

Construye un Servidor sin Express



This is my server!

And here is some more content



Explicando el proceso de renderizado

Usualmente, la aplicación web también tiene archivos css que definen el estilo para las páginas web html.

Bajo el directorio 'public', crea los siguientes directorios:

images

css

js



Bajo el lib css crea un archivo llamado 'style'

Explicando el proceso de renderizado

En style.css:

```
h1 { color: green; }
```

Y agrega un enlace a style.css en index.html:

En el elemento <head>:

```
<link rel="stylesheet" href="./css/style.css">
```

Explicando el proceso de renderizado

Ve a localhost:3000/ y ve en el registro:

```
GET request for /  
GET request for /css/style.css
```

¿Pero quién solicitó '/css/style.css'? ¡Nosotros solamente solicitamos '/'!

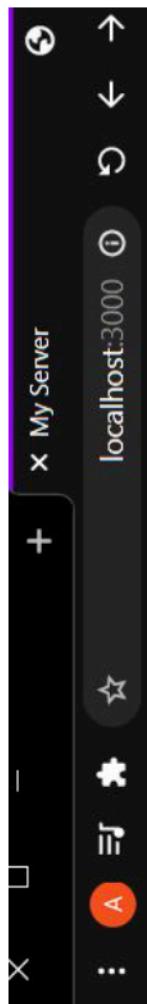
La respuesta a eso es que por cada enlace que agregamos a nuestra página web, otra solicitud es enviada.

¡Trata de agregar enlaces a imágenes o otros archivos css y mira qué sucede!

Explicando el proceso de renderizado

¿Puedes ver el problema?

Solamente manejamos la solicitud para '/' en server.js. ¡Entonces el cliente no va a recibir el archivo css!



This is my server!

And here is some more content

Manejo de archivos CSS e imágenes

Entonces, manejemos los archivos css para que el usuario obtenga el estilo de la página.

```
function getStaticPath(relativePath) {  
    return path.join(__dirname, 'public', relativePath);  
}
```

Manejo de archivos CSS e imágenes

```
http.createServer((req, res) => {
  console.log(`\$${req.method} request for ${req.url}`);
  if (req.url === "/") {
    fs.readFile("./public/index.html", "UTF-8", (err, html) =>
      res.writeHead(200, { "Content-Type": "text/html" }).end(html));
  }
  else if (req.url.match(/.css$/)) {
    const fileStream = fs.createReadStream(getStaticPath(req.url), "UTF-8");
    fileStream.pipe(res.setHeader("Content-Type", "text/css"));
  }
  else if (req.url.match(/.png$/)) {
    const fileStream = fs.createReadStream(getStaticPath(req.url), "UTF-8");
    fileStream.pipe(res.setHeader("Content-Type", "image/jpeg"));
  }
  else {
    res.writeHead(404, { "Content-Type": "text/plain" })
    .end("404 file not found");
  }
}).listen(3000);
```

Regex es un método para verificar patrones en strings.
De esta manera podemos verificar que req.url termina con '.css'

pipe es la respuesta para transmisión

También retorna imágenes.
(solamente imágenes .png en este ejemplo)

Manejo de archivos CSS e imágenes

Y finalmente, podemos ver que el título tiene color:

¡Tambien trata de agregar imágenes .img!



This is my server!

And here is some more content



wawiwa

¿Preguntas?