

HTTP Module

Creating Web Servers

Build Your Own Web Server

With just a few lines of Node.js code!

HTTP Module

The `http` module lets you create web servers and handle HTTP requests.

Key Point

The `http.Server` class **inherits from `net.Server`**, which **inherits from `EventEmitter`!**

This means creating a server creates an `EventEmitter`!

 [HTTP Module Docs](#)

Basic Server Setup

Creating a Server

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

// Create server (it's an EventEmitter!)
const server = http.createServer();

// Start listening on port 3000
server.listen(3000);
console.log('Listening on port 3000 ...');
```

The server is now running and waiting for connections!

Listening for Connection Events

Adding a Connection Listener

```
const http = require('http');
const server = http.createServer();

// Listen for 'connection' events
server.on('connection', (socket) => {
  console.log('New connection...');
});

server.listen(3000);
console.log('Listening on port 3000 ...');
```

Test it: Open your browser and go to `http://localhost:3000/`

Output:

```
milan@les2 ~ node httpvb.js
Listening on port 3000 ...
New connection...
```

✓ Better Approach: Using Callback

Handle Requests Directly

Instead of listening for ‘connection’ events, pass a callback to `createServer()`:

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

// Callback receives request and response objects
const server = http.createServer((req, res) => {
  if (req.url === '/') {
    res.write('Hello World');
    res.end();
  }

  if (req.url === '/api/courses') {
    res.write(JSON.stringify([1, 2, 3]));
    res.end();
  }
});

server.listen(3000);
console.log('Listening on port 3000 ...');
```

🧪 Testing Your Server

Try These URLs

Run the server:

```
milan@les2 ~ node httpvb2.js
Listening on port 3000 ...
```

Open in browser:

URL	Response
<code>http://localhost:3000/</code>	Hello World
<code>http://localhost:3000/api/courses</code>	[1,2,3]



Understanding the Code

Request & Response Objects

```
http.createServer((req, res) => {
  // req = Request object
  // res = Response object
});
```

Request Object (req)

- `req.url` - The URL path requested
- `req.method` - HTTP method (GET, POST, etc.)
- `req.headers` - Request headers

Response Object (res)

- `res.write()` - Write response data
- `res.end()` - End the response
- `res.statusCode` - Set HTTP status code

Building a Simple API

Example: RESTful Endpoint

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

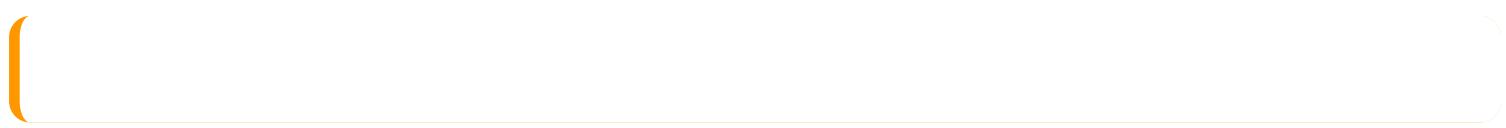
const server = http.createServer((req, res) => {
  // Set response headers
  res.writeHead(200, { 'Content-Type': 'application/json' });

  if (req.url === '/') {
    res.write(JSON.stringify({ message: 'Welcome to the API' }));
  }
  else if (req.url === '/api/users') {
    const users = [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' }
    ];
    res.write(JSON.stringify(users));
  }
  else {
    res.writeHead(404);
    res.write(JSON.stringify({ error: 'Not Found' }));
  }

  res.end();
});

server.listen(3000);
console.log('API server running on port 3000');
```

🚀 Why Use Express Instead?



The HTTP Module is Low-Level

While the `http` module works, it's quite basic:

Challenges:

- **✗** Manual URL parsing
- **✗** No built-in routing
- **✗** Verbose code for complex apps
- **✗** No middleware support

Solution: Express.js

In the next chapters, we'll use **Express.js**, which builds on top of the `http` module and makes everything easier!

🔑 Key Concepts

Concept	Explanation
<code>http.createServer()</code>	Creates an HTTP server (EventEmitter)
<code>server.listen()</code>	Start listening on a port
<code>req</code>	Request object (incoming data)
<code>res</code>	Response object (outgoing data)
<code>res.write()</code>	Write data to response

res.end()Finish and send response

Best Practices

DO

- Always call `res.end()` to finish the response
- Set appropriate status codes
- Use JSON for API responses
- Handle 404 (not found) cases
- Set proper Content-Type headers

DON'T

- Don't forget to end the response (browser hangs!)
 - Don't use http module for complex apps (use Express)
 - Don't hardcode ports (use environment variables)
 - Don't expose sensitive errors to clients
-

Chapter 2 Complete!

What You've Learned

-  Global objects vs modules
-  Creating and exporting modules
-  Using built-in modules (path, os, fs)
-  Working with events and EventEmitter
-  Building basic HTTP servers

Next: Learn about npm and package management!



Assignment

GitHub Classroom Lab

Complete the Chapter 2 lab assignment:

See Toledo for the GitHub Classroom link

Practice everything you've learned about modules!

[Course Home](#) | [Chapter 2 Home](#)

[← Previous: Events](#) | [Next Chapter: NPM →](#)