Node.js - Lab 2 - Lets make a server

# Let's make a server!

Node is not a web server, but it comes bundled with modules that let you create a web server very easily.

## Why Node makes a jolly good server

Node is single threaded and event driven. This is actually a very good architecture for a web server.

We define callback functions that listen out for events, for example network connections, file system events, API calls, etc. When the event occurs, the callback function is executed.

This means that Node does not need to spawn a new process for each connection. We can have many thousands or even millions of open connections with little impact on performance.

We also don't need to muck about with concurrency issues as much because there are no threads. We get most of the benefits of threads without the tricky bits.

## Let's create a server now.

We can create a server in Node with 6 or 7 lines of code, like so:

var http = require('http');

http.createServer(function (request, response) {

response.writeHead(200);

response.write('<h1>Hello Node!!!!</h1>\n');

response.end();

}).listen(3000);

console.log('Server running at http://localhost:3000');

This is all we need to serve a simple website. Yes, that's it.

Let's break it down a little:

### **1. Require http**

First we need to require the http module. This comes built in to Node. We require this module and save the object that the module returns in a variable. We can then use this object to create a server.

var http = require('http');

### **2. Create the server**

Now we need to create a server and set it listening.

The createServer function receives a callback function. This is the function that will be called when a network request comes in.

http.createServer(function (request, response) {

// You have access to the request and response objects here.

}).listen(3000);

### **3. Write to the response**

The callback function receives two objects, which we typically call request and response (or req and res if you prefer)

Whenever someone hits our website, Our callback function will receive request and response objects. We can write whatever we like to the response head and body, and finally return the result:

response.writeHead(200); // write a 200 OK header

response.write('<h1>Hello Node!</h1>

'); // write to the body

response.end(); // return the response to the user

We call response.end() to return the response. If you forget this your server will never return any content. This is by design as it allows us to create long lasting connections that only return a result when something interesting happens.

## Exercise - Create a server

Create a file containing the code above.

In a console, navigate to the directory in which you have saved your file. Start the server with:

node app.js

Visit [http://localhost:3000](http://localhost:3000/) to ensure you have achieved success.

### **Extension**

* Modify your server to make it return a string of your choosing.
* Modify your server to make it listen to port 5000.

## Request and Response objects

You will remember from the previous exercise that our callback function receives two objects which we commonly call request and response.

The request object contains information about the request. The response object is where we write our server response.

## Exercise - Log the request

Modify your server. Use console.log() to output the entire request object to the console. Have a look around inside, everything is there.

Have a look through the request object. Now try to console.log just the current URL.

Now modify your code. Try to write the current URL as a string to the response. If I visit <http://localhost:3000/nacktschnecke> I should see a web page containing the word nacktschnecke, ideally wrapped in an h1 tag.

## Exercise - A simple router

Given that we can gain access to the URL, write a simple 2 page website that responds to <http://localhost:3000/nacktschnecke> and <http://localhost:3000/about> and serves content appropriately. Do this using if, else if and else.

If no route matches, have your server return a 404 file not found status code and page.